ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інформаційно-аналітична система інтелектуального моніторингу та прогнозування енергозабезпечення на базі IoT-сумісних інверторів і графіків відключення світла»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Технології цифрового розвитку»

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

_____ Нікіта БОЙЧЕНКО
(підпис)

Виконав: здобувач(ка) вищої освіти групи ТЦР-41
Нікіта БОЙЧЕНКО

Керівник: Андрій АРОНОВ
кандидат технічних наук

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бойченку Нікіті Олеговичу

1. Тема кваліфікаційної роботи: «Інформаційно-аналітична система інтелектуального моніторингу та прогнозування енергозабезпечення на базі IoT-сумісних інверторів і графіків відключення світла»
керівник кваліфікаційної роботи кандидат технічних наук Андрій АРОНОВ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи « 19 » травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: офіційна документація Python, Aiogram, SQLite, SQLAlchemy і Termux, технічні матеріали виробника інверторного обладнання Delongtop та зовнішнього моніторингового середовища mini-ems, офіційні матеріали ДТЕК щодо графіків відключень електроенергії, довідкові матеріали сервісу bezsvitla.com.ua.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідити предметну область моніторингу електропостачання та резервного живлення в побутових умовах України, визначити проблеми

ручного контролю інвертора і графіків відключень, а також сформувати функціональні очікування від програмної системи.

2. Проаналізувати існуючі підходи та джерела даних для отримання графіків відключень і моніторингу інверторного обладнання, зокрема офіційні матеріали ДТЕК, сервіс bezsvitla.com.ua, веб-сервіс mini-ems та можливі підходи до побудови користувацького інтерфейсу.

3. Спроектувати інформаційно-аналітичну систему у формі Telegram-бота: визначити вимоги до системи, моделі взаємодії користувачів, архітектуру програмної системи, структуру локального сховища даних і основні користувацькі сценарії.

4. Реалізувати та описати функціонування Telegram-бота, включно з Telegram-взаємодією, моніторингом інвертора, системою сповіщень, підсистемою графіків відключень, адміністративними можливостями, AI-функціями та особливостями розгортання системи.

5. Перелік графічного матеріалу: *презентація*
   1. Мета, об'єкт та предмет дослідження
   2. Задачі кваліфікаційної роботи
   3. Аналіз існуючих підходів
   4. Вимоги до системи
   5. Програмні засоби реалізації
   6. Діаграма варіантів використання системи
   7. Архітектура програмної системи
   8. Основні алгоритмічні сценарії
   9. Тестування
   10. Екранні форми: Telegram-взаємодія
   11. Екранні форми: повідомлення системи
   12. Апробація результатів дослідження та висновки

6. Дата видачі завдання « 20 » лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз існуючої міні-системи та огляд аналогічних рішень	21.02–08.03.2026	
2	Визначення напрямків удосконалення існуючої міні-системи	09.03–22.03.2026	
3	Проєктування архітектури удосконаленої системи та структури бази даних	23.03–05.04.2026	

4	Удосконалення програмного забезпечення: міграція на асинхронну модель	06.04–19.04.2026	
5	Налагодження удосконаленої системи	20.04–26.04.2026	
6	Оформлення роботи: вступ, висновки, реферат	27.04–02.05.2026	
7	Розробка демонстраційних матеріалів для презентації	03.05–06.05.2026	
8	Попередній захист роботи	07.05–08.05.2026	

Здобувач(ка) вищої освіти _____ Нікіта БОЙЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Андрій АРОНОВ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 67 стор., 12 табл., 17 рис., 21 джерел.

Мета роботи – спрощення взаємодії побутового користувача з даними про електропостачання, резервне живлення, графіки відключень і параметри інвертора, за допомогою розробки аналітичної системи у вигляді Telegram-бота.

Об'єкт дослідження – процес інформаційного супроводу побутового користувача під час контролю стану електропостачання, резервного живлення та графіків відключень електроенергії.

Предмет дослідження – методи і програмні засоби побудови системи, що об'єднує телеметрію інвертора та публічні графіки відключень у єдиному побутовому інтерфейсі.

Короткий зміст роботи: У роботі проаналізовано предметну область побутового моніторингу електропостачання та резервного живлення, виокремлено функціональні очікування побутового користувача до інформаційно-аналітичної системи моніторингу, критично розглянуто зовнішні джерела даних – офіційні канали публікації графіків ДТЕК, неофіційний сервіс агрегації bezsvitla.com.ua та зовнішнє моніторингове середовище інвертора mini-ems, обґрунтовано вибір технологічного стеку на основі мови Python, бібліотеки aiogram, реляційного сховища SQLite та об'єктно-реляційного відобразника SQLAlchemy. Сформульовано вимоги до системи, моделі ролей користувачів, спроєктовано чотирирівневу архітектуру, структуру локального сховища даних і основні алгоритмічні сценарії. Реалізовано програмну систему у вигляді Telegram-бота SvetaNeBudit_bot з підтримкою багатокористувацького сценарію, дедуплікації фізичних інверторів, фонового опитування зовнішніх джерел, формування сповіщень про зміну стану зовнішньої мережі та про аномальні діапазони напруги з порогами 260В і 170В, а також послідовності резервних сценаріїв отримання графіка погодинних відключень.

КЛЮЧОВІ СЛОВА: МОНІТОРИНГ ЕЛЕКТРОПОСТАЧАННЯ, ІНВЕРТОР, РЕЗЕРВНЕ ЖИВЛЕННЯ, ГРАФІКИ ВІДКЛЮЧЕНЬ, ІНФОРМАЦІЙНО-АНАЛІТИЧНА СИСТЕМА, TELEGRAM-БОТ, PYTHON, AIOGRAM, SQLITE, TERMUX.

ЗМІСТ

ВСТУП

Актуальність теми. Упродовж останніх років в Україні принципово змінилася роль електропостачання у побутовому середовищі: з мовчазного фонового ресурсу воно перетворилося на ресурс змінної доступності, параметри якого потребують систематичного контролю. У межах типового домогосподарства одночасно виникли три взаємопов'язані прикладні задачі – стеження за фактом наявності або відсутності зовнішнього живлення, узгодження побутової активності з публічними графіками погодинних відключень та контроль якості напруги у моменти присутності мережі, – до яких додалося повсюдне поширення інверторних систем із акумуляторними батареями, що одночасно забезпечують безперебійність живлення критичних навантажень і виступають джерелом телеметрії.

Попри формальну доступність кожного з наведених джерел інформації, у типовому сценарії робота з ними лишається ручною. Перевірка стану інвертора через штатне зовнішнє моніторингове середовище виробника вимагає окремої свідомої дії, автентифікації і самостійного розуміння технічних показників. Офіційні ресурси публікації графіків відключень побудовані навколо адресного сценарію, непридатного для багатократного автоматичного відстеження; неофіційні веб-агрегатори, що переподають оприлюднену інформацію у зручнішому вигляді, також вимагають окремого свідомого звернення. Розпорошеність джерел стає самостійним фактором незручності, а ручний характер моніторингу за визначенням не забезпечує оперативного інформування про зміну стану електропостачання та про небезпечні відхилення параметрів мережі. Актуальність теми підсилюється прикладним інцидентом, зафіксованим у домогосподарстві під час підготовки проекту: перевищення напруги понад 310В не супроводжувалося практично корисним побутовим попередженням з боку зовнішнього середовища. Тому така ситуація обґрунтовує потребу у побутово орієнтованій інформаційно-аналітичній системі, що об'єднує телеметрію інвертора та графіки погодинних відключень у єдиному інтерфейсі, придатному до роботи у мобільному середовищі.

Мета роботи – спрощення взаємодії побутового користувача з даними про електропостачання, резервне живлення, графіки відключень і параметри інвертора, за допомогою розробки аналітичної системи у вигляді Telegram-бота.

Для досягнення мети, у бакалаврській роботі успішно виконано наступні завдання:

1. Дослідити особливості побутового контролю електропостачання та резервного живлення в умовах України.
2. Проаналізувати проблеми ручного моніторингу інвертора і графіків відключень та визначити функціональні очікування від майбутньої системи.
3. Проаналізувати офіційні канали ДТЕК[1], сервіс bezsvitla.com.ua[4] та зовнішнє середовище mini-ems[8] як джерела даних для програмної системи.
4. Обґрунтувати вибір архітектурного підходу, користувацького інтерфейсу та технологічного стеку для реалізації системи.
5. Створити вимоги до системи, моделі взаємодії користувачів, архітектуру програмної системи, структуру сховища даних і основні сценарії її роботи.
6. Реалізувати Telegram-бота, що об’єднує моніторинг інвертора, підсистему графіків відключень, систему сповіщень, адміністративні функції та допоміжні AI-можливості.

Об’єкт дослідження – процес інформаційного супроводу побутового користувача під час контролю стану електропостачання, резервного живлення та графіків відключень електроенергії.

Предмет дослідження – методи і програмні засоби побудови системи, що об'єднує телеметрію інвертора та публічні графіки відключень у єдиному побутовому інтерфейсі.

Методи дослідження. У роботі використано методи аналізу предметної області, порівняльного аналізу існуючих підходів і технологічних рішень, методи проектування інформаційних систем і сховищ даних, методи моделювання користувацьких сценаріїв та архітектури програмного забезпечення, а також практичні методи реалізації, інтеграції й налагодження Telegram-ботів, веб-парсингу, обробки даних та організації автоматичних сповіщень.

Наукова новизна одержаних результатів. У запропонованому підході до побудови інформаційно-аналітичної системи, яка в межах одного Telegram-інтерфейсу інтегрує зовнішнє моніторингове середовище mini-ems[8], персоналізоване отримання графіків відключень та механізми подійного інформування про стан електроживлення. Додаткову новизну становить реалізація багатокористувацької моделі доступу до одного фізичного інвертора, буферизованого надання статусу без прямого зовнішнього запиту при кожному зверненні та адаптація всієї системи до мобільного середовища Termux[18].

Практична значущість одержаних результатів. Створення програмної системи, орієнтованої на реальні побутові сценарії використання резервного живлення. Розроблене рішення дає змогу користувачеві в одному інтерфейсі отримувати графік відключень за своєю групою, контролювати стан інвертора, своєчасно отримувати повідомлення про зміну стану зовнішньої мережі та користуватися зручним інструментом повсякденного інформаційного супроводу під час перебоїв електропостачання.

Апробація результатів та публікації. Основні положення і результати бакалаврської роботи доповідались на всеукраїнських науково-технічних конференціях:

1. Бойченко Н.О., Аронов А.О. Особливості персоналізованого надання графіків відключень світла у телеграм-боті для моніторингу енергозабезпечення. // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. – К.: ДУІКТ, 2026, (подано до друку).
2. Бойченко Н.О., Аронов А.О. Спрощення взаємодії моніторингу інвертора через телеграм-бот з автоматичними сповіщеннями та зрозумілим поданням даних. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. – К.: ДУІКТ, 2026, (подано до друку).

РОЗДІЛ 1. ПРЕДМЕТНА ОБЛАСТЬ МОНІТОРИНГУ ЕЛЕКТРОПОСТАЧАННЯ ТА РЕЗЕРВНОГО ЖИВЛЕННЯ

1.1. Особливості контролю електропостачання в побутових умовах України

Сучасний побутовий споживач електроенергії в Україні перебуває в принципово іншій ситуації, ніж той, для якого проектувалися типові побутові електромережі. Електропостачання перестало бути послугою, що мовчазно існує у фоні: воно стало ресурсом, доступність і якість якого змінюються кілька разів на добу, потребуючи від користувача систематичного контролю. Цей контроль охоплює щонайменше три аспекти: факт наявності або відсутності напруги в зовнішній мережі, відповідність фактичного стану заздалегідь оголошеному графіку погодинних відключень і дотримання нормованих параметрів напруги в моменти, коли мережа фізично присутня. У сукупності ці аспекти формують самостійну прикладну задачу побутового моніторингу, для якої традиційні засоби побутової автоматики не пристосовані.

Першою і найочевиднішою особливістю є нестабільність мережі. Для значної частини домогосподарств зникнення зовнішнього електроживлення є не аварійною подією, а очікуваним повторюваним станом, тривалість якого обчислюється годинами. Зворотний бік цієї ж особливості – фрагментованість періодів наявності світла, коли вікна доступного електроживлення доводиться планомірно використовувати для заряджання резервних джерел, побутової техніки і пристроїв зв'язку. Така фрагментованість суттєво підвищує цінність будь-якого механізму, який автоматично фіксує момент появи або зникнення мережі, оскільки користувач здебільшого фізично не перебуває біля точки споживання у потрібну мить.

Другою особливістю є наявність публічних графіків погодинних відключень, які стали де-факто інструментом планування побутового життя. Офіційні енергопостачальні компанії публікують такі графіки на власних інформаційних ресурсах, орієнтуючи користувача переважно на адресний сценарій – введення

адреси для отримання індивідуальної інформації. Поряд із цим у публічному просторі функціонують неофіційні агрегатори, які автоматично відстежують публікації енергокомпаній, переоформлюють їх у компактний вигляд за регіонами та чергами і публікують у власних веб-ресурсах або Telegram-каналах. У контексті побутового моніторингу принципово важливо розрізняти ці два рівні: офіційні канали є першоджерелом, тоді як агрегатори лише переподають отриману інформацію у зручнішій формі. Незалежно від обраного джерела, користувач щоразу повинен самостійно знайти потрібний регіон, потім потрібну чергу і лише тоді зіставити її стан із поточним часом.

Третьою особливістю є якість напруги в моменти, коли зовнішня мережа доступна. Перехідні режими, що супроводжують масові комутації після відновлення живлення в районі, можуть призводити як до перенапруги, так і до зниженої напруги відносно номінального значення 220В. Ці відхилення безпосередньо загрожують побутовій техніці і резервному обладнанню, але візуально вони непомітні, тому що мережа формально «є». Під час проектування програмного продукту фіксується реальний інцидент, коли напруга в точці домогосподарства перевищила 310В, тоді як зовнішнє моніторингове середовище виробника інверторного обладнання не сформувало для побутового користувача практично корисного попередження. Цей випадок ілюструє, що побутовий контроль електропостачання не зводиться до бінарного стану «світло є / світла немає», а вимагає окремих порогових тривог, орієнтованих не на технічні діапазони обладнання, а на безпеку домогосподарства.

Четвертою особливістю, що логічно випливає з попередніх, є зростання ролі резервного живлення. Інверторні системи разом з акумуляторними батареями стали типовим елементом побутової інфраструктури: вони підхоплюють критичні навантаження в моменти зникнення мережі і повертаються до режиму заряджання, щойно живлення відновлюється. Інвертор у такому використанні є не просто перетворювачем напруги, а локальним джерелом телеметрії: він відомий стан зовнішньої мережі, рівень заряду батареї, поточну вихідну потужність, температурні показники та режим роботи. Більшість сучасних інверторів

відповідного класу штатно підтримують підключення WiFi-модуля та передачу зазначеної телеметрії у зовнішнє моніторингове середовище виробника. Для розглянутого в роботі обладнання таким середовищем є сервіс mini-ems[8], через який поступає вся практично доступна інформація про роботу пристрою.

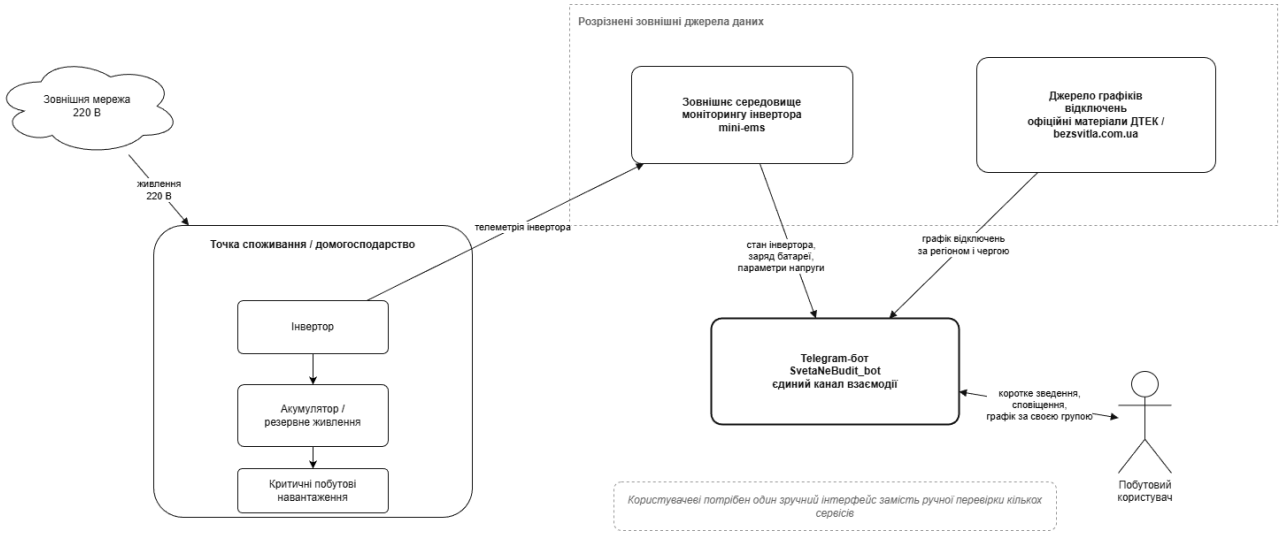


Рис. 1.1 Узагальнена схема зведення різних джерел інформації у єдиному каналі взаємодії

1.2. Проблеми ручного моніторингу інвертора та графіків відключень

Зазначений у попередньому підрозділі набір джерел даних формально доступний широкому колу користувачів, однак спосіб взаємодії з ним за замовчуванням є ручним. Ручний характер моніторингу породжує ряд практичних проблем, які в сукупності і визначають мотивацію до створення окремої інформаційно-аналітичної надбудови над цими джерелами.

Перша група проблем пов'язана з ручною перевіркою стану інвертора через зовнішнє моніторингове середовище виробника. Для отримання поточних показників користувач має самостійно перейти у відповідний веб-сервіс або мобільний застосунок виробника, виконати автентифікацію, обрати об'єкт спостереження та потрібний пристрій і лише після цього отримати екран із технічними параметрами. Цей сценарій орієнтований передусім на самостійну

перевірку та потребує технічних знань: він припускає періодичну, але не безперервну взаємодію, не передбачає миттєвої реакції на зміну стану та оперує термінологією і зведеннями, природними для технічного спеціаліста, а не для побутового користувача.

Окрему складність ручного сценарію створює необхідність кожного разу поновлювати сесію з зовнішнім моніторинговим сервісом. У повсякденному використанні це означає повторні автентифікації, ручний перехід між розділами інтерфейсу і подвійну роботу при наявності в одному домогосподарстві кількох зацікавлених осіб: кожна з них має самостійно мати доступ і самостійно виконувати ті самі дії.

Друга група проблем стосується ручної роботи з графіками погодинних відключень. Хоч офіційні енергопостачальні компанії і неофіційні агрегатори надають публічно доступні засоби перегляду, користувач за кожним зверненням самостійно проходить однотипну послідовність кроків: відкриває сторінку, обирає область, обирає чергу, орієнтується в добовій сітці і самостійно зіставляє її з поточним часом. Якщо домогосподарство цікавить стан не лише на сьогодні, але й на наступну добу, цей цикл доводиться повторювати окремо. Ручна перевірка ускладнюється і тим, що структура поданих даних час від часу змінюється: переоформлення сторінки на джерелі або тимчасова відсутність опублікованих даних на поточний день вимагає від користувача самостійних рішень про те, чи вважати сторінку «порожньою» помилкою, чи орієнтуватися за раніше відомим графіком. Для непідготовленого користувача такий сценарій є психологічно стомливим, бо в період підвищеного навантаження на енергосистему перевірка може повторюватися кілька разів на добу.

Третя і фактично об'єднувальна проблема – відсутність єдиного інтерфейсу. Інверторне обладнання й публічні графіки відключень походять з різних доменів, належать різним постачальникам інформації і подаються через різні канали. Для домогосподарства, проте, вони описують одну й ту саму прикладну реальність – стан електропостачання в конкретній точці. Розпорошення цих даних по окремих сервісах змушує користувача утримувати в пам'яті кілька наборів облікових

даних, тримати відкритими кілька вкладок або застосунків і самостійно зводити між собою отриману інформацію.

Четверта проблема має технічну природу, але побутові наслідки. Ручний моніторинг за визначенням не дає оперативних сповіщень. Коли в зовнішній мережі зникає напруга, користувач отримує цю інформацію лише в момент, коли сам зазирає у відповідний інтерфейс або фізично помічає зміну освітлення в приміщенні. Так само ручний сценарій не повідомляє про повернення живлення, унаслідок чого вікна для зарядження резерву можуть пропускатися.

Таблиця 1.1

Основні проблеми ручного моніторингу та їх наслідки

Тип проблеми	Опис	Поведінка користувача	Небажаний наслідок
Ручна перевірка стану інвертора	Окрема свідома дія: перехід у веб-сервіс або застосунок виробника, автентифікація, вибір об'єкта.	Перевіряє стан лише коли про це згадав; не отримує сповіщень між зверненнями.	Зміни стану між перевірками залишаються непоміченими.
Ручна перевірка графіка	Повторний вибір області, черги, зіставлення добової сітки з поточним часом.	За кожним зверненням повторює той самий ланцюг кроків; не відрізняє тимчасову відсутність публікації від реальної проблеми.	Хибні рішення щодо планування побутової активності та дратівлива повторюваність дій.
Відсутність єдиного інтерфейсу	Розпорошення інверторного і графічного доменів між різними сервісами, обліковими даними, додатками.	Утримує в пам'яті кілька наборів доступу, самостійно зводить дані.	Кратне зростання трудомісткості у багатокористувацькому домогосподарстві.
Відсутність оперативних сповіщень	Жоден ручний сценарій не сповіщає про появу/зникнення живлення і про небезпечні відхилення напруги.	Пропускає вікна заряджання резерву; не встигає відреагувати на перенапругу.	Ризик для побутової техніки, фактична неможливість оперативної реакції на аварійні стани.

У випадку небезпечних відхилень напруги ситуація стає критичнішою: ручна перевірка теоретично здатна зафіксувати такий стан, але практично майже не дає шансу зреагувати, оскільки користувач рідко відкриває моніторингове середовище саме у момент аварійного режиму. Згаданий вище реальний інцидент із напругою понад 310В показує, що навіть наявність технічного джерела даних сама собою не розв'язує задачі побутової безпеки: для цього потрібен механізм виштовхування інформації до користувача за заздалегідь установленими порогами, а не лише її пасивне зберігання та подання за запитом.

Перелічені проблеми не є дефектами окремо взятого джерела. Кожне з них: і зовнішнє моніторингове середовище інвертора, і офіційні та неофіційні канали публікації графіків – виконує своє вузьке завдання, для якого було спроектоване. Проблема виникає на стику: побутовому домогосподарству потрібен синтез цих джерел у форматі, який не передбачений жодним із них окремо. Саме цей розрив і визначає предметну нішу для інформаційно-аналітичної системи моніторингу, функціональні очікування від якої окреслено в наступному підрозділі.

1.3. Функціональні очікування від системи моніторингу

Розглянуті вище особливості предметної області та виявлені проблеми ручного моніторингу дозволяють сформулювати функціональні очікування від побутової інформаційно-аналітичної системи моніторингу електропостачання. У межах цього підрозділу очікування подано саме як вимоги предметної області, без переходу до архітектурних рішень, реалізаційних деталей чи опису технологічного стеку.

Базовою категорією очікувань є оперативне інформування про зміну стану зовнішньої мережі. Користувач має дізнаватися про зникнення напруги і про її відновлення без необхідності самостійно ініціювати перевірку. Доставка такого повідомлення повинна відбуватися через канал, який природно вбудований у повсякденне користування смартфоном, тобто через звичні засоби push-сповіщень. Зміст повідомлення має бути коротким і однозначним. Така постановка означає, що

система не лише пасивно відображає стан, але й самостійно відстежує переходи між станами, фіксуючи їх як події, гідні окремого сповіщення користувачу.

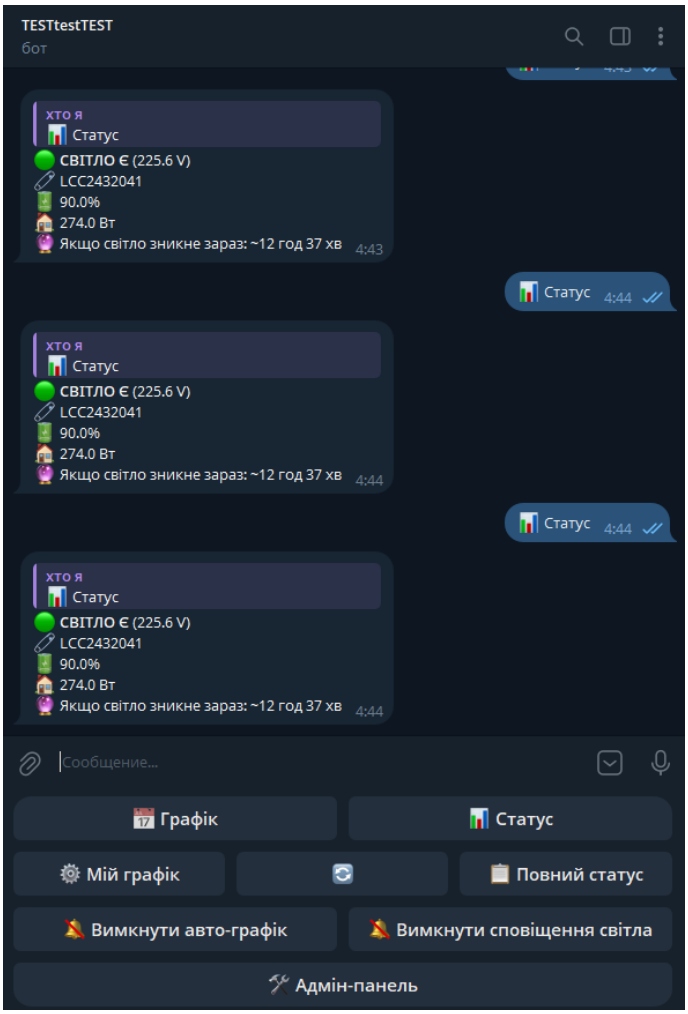


Рис. 1.2 Приклад очікуваного вигляду короткого зведення стану в Telegram-інтерфейсі

Другою категорією очікувань є тривоги, прив'язані не до факту наявності або відсутності живлення, а до його якості. Беручи до уваги практику побутових інцидентів, описаних у попередніх підрозділах, доцільно виділити два окремих порогових сповіщення: про небезпечно високу напругу та про небезпечно низьку напругу. Пороги для цих сповіщень мають визначатися не стільки технічними діапазонами обладнання, скільки безпекою побутової електроніки, тому в межах роботи прийнято орієнтири понад 260 В для верхнього і нижче 170 В для нижнього порогу. Принципово важливо, щоб такі тривоги мали характер крайових подій:

повідомлення доставляється у момент входу в небезпечний стан, але не повторюється у кожному циклі моніторингу, поки стан зберігається. Така поведінка зменшує ризик «втоми від сповіщень», коли надмірна частота повідомлень знецінює їх для користувача.

Третя категорія очікувань стосується перегляду поточного стану на запит. Користувач має мати можливість швидко отримати коротке зведення без виконання повної повторної автентифікації у зовнішньому моніторинговому середовищі і без помітної затримки. Зміст такого зведення повинен бути обмежений лише найбільш практично цінною інформацією: чи присутня зовнішня мережа, який рівень заряду резервного джерела, яке поточне навантаження. Усі додаткові технічні поля, корисні переважно сервісному інженеру, у короткому зведенні не висвітлюються, щоб не розпорошувати увагу користувача.

Четверта категорія очікувань охоплює роботу з графіками погодинних відключень. Користувач має мати можливість самостійно вказати свою область і чергу один раз і надалі отримувати персоналізовану інформацію без повторного вибору під час кожного звернення. Окремою функціональною вимогою є автоматичне інформування про появу або зміну графіка на наступний день, якщо така опція ввімкнена користувачем. У моменти, коли публічне джерело графіків тимчасово недоступне, або коли його структура зазнала змін, або коли на поточний день дані ще не опубліковано, від системи очікується не аварійна відмова, а прозоре повідомлення користувачу про природу проблеми та, за наявності, показ найбільш свіжого збереженого варіанту з відповідною позначкою. Ця сукупність очікувань фактично визначає вимогу багатоступеневої логіки запасних варіантів подання графіка.

П’ята категорія очікувань пов'язана з охопленням аудиторії. Не всі користувачі побутового моніторингового сервісу володіють інверторним обладнанням, але навіть у такому разі вони залишаються зацікавленими у централізованому отриманні графіків відключень. З цього випливає вимога окремого режиму використання для користувачів без інвертора, у межах якого функціональність обмежується саме роботою з графіками і не передбачає введення

облікових даних до зовнішнього моніторингового середовища. Такий режим суттєво розширює аудиторію, для якої система є практично корисною, і водночас спрощує сценарій початкової взаємодії.

Шоста категорія очікувань стосується багатокористувацького використання в межах одного домогосподарства або кола довірених осіб. На один фізичний інвертор зазвичай претендують щонайменше кілька членів родини, які бажають отримувати сповіщення і мати доступ до зведень. Очікувано, що система розрізняє роль власника інвертора, який вводить облікові дані до зовнішнього моніторингового середовища одноразово, і роль приєднаних спостерігачів, які користуються вже зареєстрованим обладнанням без повторного введення секретних даних. Таке розподілення ролей не лише є зручною для користувачів, але й знижує надлишкове навантаження на зовнішнє моніторингове середовище, оскільки уникає створення кількох однакових, але незалежних аккаунтів для одного й того самого фізичного пристрою.

Окремою системною вимогою, що випливає з ролей власника і приєднаного спостерігача, є необхідність керування цими зв'язками: відключення спостерігачів, заміни прив'язаного інвертора без втрати раніше накопиченої історії, контрольоване припинення спостереження в разі зміни обладнання чи власника. Усі ці операції повинні виконуватися з боку користувача нескладно і не вимагати супроводу адміністратора.

Поруч із базовими функціональними очікуваннями доцільно виокремити допоміжні. Сюди належить, зокрема, інтерактивний помічник, який за поточним контекстом моніторингу здатен сформувати відповідь у вільній формі на запитання користувача побутовою мовою. Такий помічник у межах роботи розглядається саме як допоміжний компонент, що використовує наявний контекст моніторингу, але не утворює смислового центра системи. Так само до допоміжних належить адміністративний інтерфейс, який забезпечує службовий перегляд агрегованих відомостей про користувачів, телеметрію та активність, але не призначений для звичайного побутового сценарію використання.

Окремо слід зафіксувати очікування щодо середовища функціонування системи. Із попередніх підрозділів випливає, що цільовим середовищем взаємодії для побутового користувача є смартфон і месенджер, а не повноцінний веб-портал. Це накладає вимогу до системи бути придатною до тривалого автономного функціонування на одиничній мобільній платформі, а не лише до розгортання в повноцінному серверному оточенні. Деталізація такої вимоги, у тому числі вибір технологічного стеку та обмеження мобільного середовища, виходить за межі цього розділу і розкривається далі.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ І ВИБІР ТЕХНОЛОГІЙ

2.1. Аналіз офіційних каналів ДТЕК і сервісу bezsvitla.com.ua як джерел графіків відключень

З огляду на сформульовані в попередньому розділі функціональні очікування, центральним джерелом інформації про погодинні відключення електроенергії в межах розглядуваного проекту є публічні графіки, оприлюднювані енергопостачальними компаніями. Для коректного академічного опису обраного рішення необхідно чітко розрізняти два принципово різні рівні цього джерела: офіційні канали публікації, які належать енергокомпанії і виступають первинним видавцем інформації, і неофіційні агрегатори, які лише переподають отримані дані у власному форматі. Змішування цих рівнів є типовою помилкою побутового користувача, але неприпустиме в межах кваліфікаційної роботи, де природа джерела безпосередньо впливає на оцінку надійності системи в цілому.

Офіційний рівень публікації для географічного контексту, актуального в роботі, представлений двома окремими ресурсами енергопостачальної компанії: ресурсом для міста Києва – ДТЕК Київські електромережі[2] і ресурсом для Київської області – ДТЕК Київські регіональні електромережі[3]. Обидва ресурси належать одній групі компаній, але є самостійними з точки зору доменного імені, географічного покриття та практичної логіки оприлюднення графіків. Поряд з основним веб-сайтом енергокомпанія підтримує власні офіційні чат-боти у месенджерах Viber і Telegram, які офіційно рекомендуються користувачам як альтернативний канал доступу до інформації, особливо в години пікового навантаження на сайт. Але у таких джерелах оприлюднюють не персоніфіковані графіки, а саме всі черги відключень в одному повідомленні. Принципово важливо, що публічна офіційна риторика енергокомпанії на власних сторінках кваліфікує оприлюднені графіки як орієнтовні, тобто такі, що відображають графіки можливих відключень і можуть змінюватися протягом доби.

Водночас офіційні матеріали ДТЕК дають ще одну важливу для проекту рамку: самі графіки є ситуативним інформаційним шаром, а не постійно діючим сервісом з гарантованою щоденною наявністю розкладу. На офіційних сторінках прямо вказується, що графіки діють у ті дні та години, коли за командою НЕК Укренерго[21] застосовуються стабілізаційні відключення. Під час екстрених відключень, навпаки, компанія прямо зазначає, що графіки не діють. Це означає, що у періоди нормального функціонування енергосистеми або у періоди виключно екстрених заходів сам контур роботи з погодинними графіками може не мати актуального наповнення.

Таблиця 2.1

Офіційні ресурси ДТЕК, релевантні темі роботи

Ресурс	Географічна зона	Основний сценарій доступу	Характер графіків	Офіційні чат-боти
ДТЕК Київські електромережі (dtek-kem.com.ua)	Місто Київ	Адресний пошук (введення адреси абонента)	Орієнтовні графіки можливих відключень; діють лише за командою Укренерго	Так (Viber, Telegram)
ДТЕК Київські регіональні електромережі (dtek-krem.com.ua)	Київська область	Адресний пошук (введення адреси абонента)	Орієнтовні графіки можливих відключень; у періоди екстрених відключень не діють	Так (Viber, Telegram)

Спосіб взаємодії з офіційним веб-ресурсом за замовчуванням побудовано навколо адресного сценарію. Користувач вводить населений пункт, вулицю та номер будинку, після чого отримує індивідуальний результат для введеної адреси. Цей сценарій раціональний з точки зору первинного видавця інформації: він мінімізує неоднозначність прив'язки до конкретного абонента, але водночас має суттєві наслідки для побудови автоматизованої системи. Адресний пошук є інтерактивним і не передбачає прямого програмного запиту до структурованого

набору даних усього регіону; для багатократного автоматичного відстеження стану він непридатний без значних додаткових зусиль зі сторони розробника, а також може створювати небажане навантаження на офіційний ресурс. Альтернативний канал у формі офіційного чат-бота месенджера є зручним для користувача, але, по суті, теж побудований на діалоговому адресному сценарії і не виступає публічним структурованим джерелом для зовнішньої системи моніторингу.

У цих умовах зростає роль неофіційних агрегаторів, які поверх офіційно оприлюдненої інформації будують власний шар представлення.

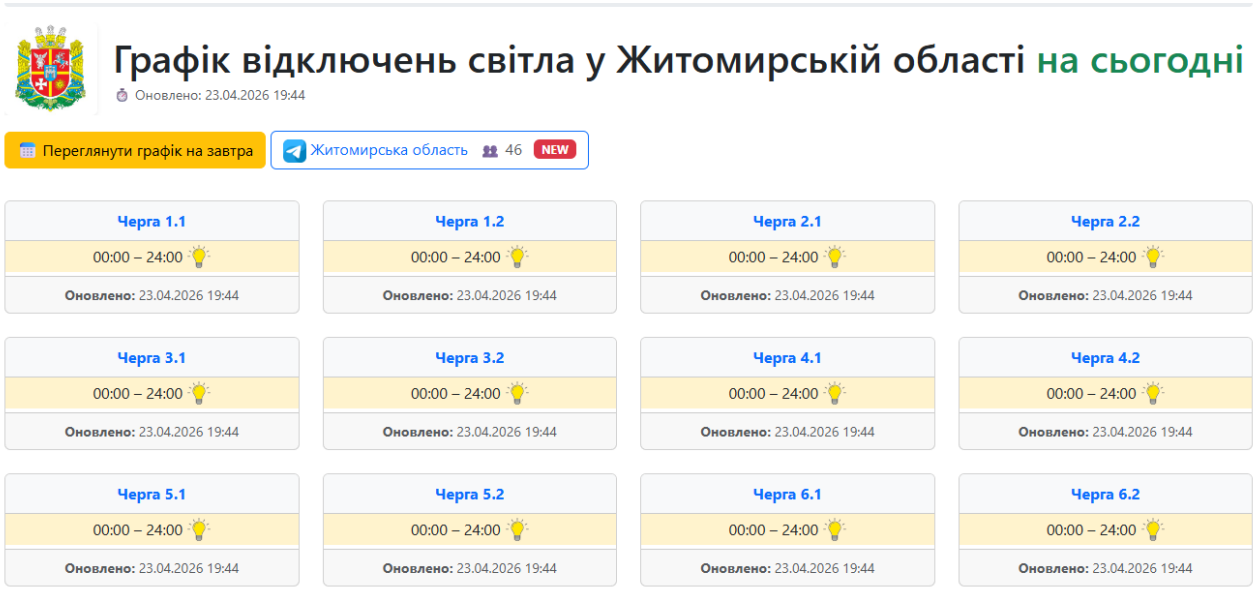


Рис. 2.1 Типова сторінка графіка відключень на сервісі bezsvitla.com.ua

Одним з таких ресурсів, релевантних задачі роботи, є веб-сервіс bezsvitla.com.ua[4]. На сторінці «Про нас» цей сервіс прямо позиціонує себе як автоматизований інструмент моніторингу публічних джерел енергопостачальних компаній, що збирає, обробляє і переоформлює оприлюднені графіки в єдиний компактний розклад, доступний через власний веб-інтерфейс і власні Telegram-канали. На регіональних сторінках сервісу, зокрема для Києва, окремо зазначено, що дані отримано з офіційного сайту ДТЕК Київські електромережі[2], а представлені графіки мають орієнтовний характер. Сервіс прямо відмежовує свій

статус від офіційного: у його власній публічній заяві зазначено, що він не є офіційним джерелом і не представляє державні установи чи енергетичні компанії.

Практичне обґрунтування використання саме bezsvitla.com.ua[5] як джерела для розглядуваної інформаційно-аналітичної системи полягає в кількох взаємопов'язаних факторах. По-перше, представлення графіка на агрегаторі є структурованим за регіоном і чергою без необхідності багатоступеневого адресного діалогу для кожного запиту, що відповідає очікуваній моделі персоналізованої підписки користувача за регіоном і чергою, сформульованій у підрозділі 1.3. По-друге, регулярне оновлення представлених на агрегаторі графіків та наявність окремих сторінок на регіон і чергу дозволяють реалізувати стійку фонову перевірку без імітації людської адресної взаємодії з офіційним сайтом. По-третє, агрегатор підтримує власне публічне оприлюднення в Telegram, що додатково підтверджує практичну стабільність формату подання й мінімальну ймовірність несумісних змін представлення в короткому горизонті спостереження.

2.2. Аналіз веб-сервісу mini-ems як зовнішнього середовища моніторингу інвертора

Друге зовнішнє джерело, на яке спирається розглядувана інформаційно-аналітична система, належить до інверторної предметної області. Йдеться про веб-сервіс mini-ems[8], через який стає доступною телеметрія інверторного обладнання користувача. Як і у випадку з джерелами графіків відключень, академічно коректне формулювання вимагає чіткого розмежування внутрішнього і зовнішнього: mini-ems[8] не є внутрішнім модулем розробленої системи і не входить до її програмної реалізації. Він розглядається як зовнішнє моніторингове середовище виробника інверторного обладнання, на яке система спирається як на джерело даних, не замінюючи при цьому штатний сервісний сценарій налаштування обладнання, передбачений виробником.

Підставою для опису саме у такому статусі є два рівні підтвердження. Перший рівень – публічні матеріали виробника. Офіційні матеріали Delongtop[6]

для відповідної лінійки інверторно-акумуляторних систем послідовно зазначають наявність опціонального WiFi-модуля віддаленого моніторингу, можливість моніторити стан системи через мобільний застосунок або веб-сайт, а також підтримку каналів RS485 / CAN / WiFi для комунікації обладнання із зовнішніми системами. Користувацькі настанови виробника описують підключення WiFi/GPRS plug-and-play модуля моніторингу і доступ до інформації про стан установки в мобільному та веб-інтерфейсах. Тим самим зовнішнє моніторингове середовище є штатною передбаченою функціональністю обладнання, а не сторонньою надбудовою.

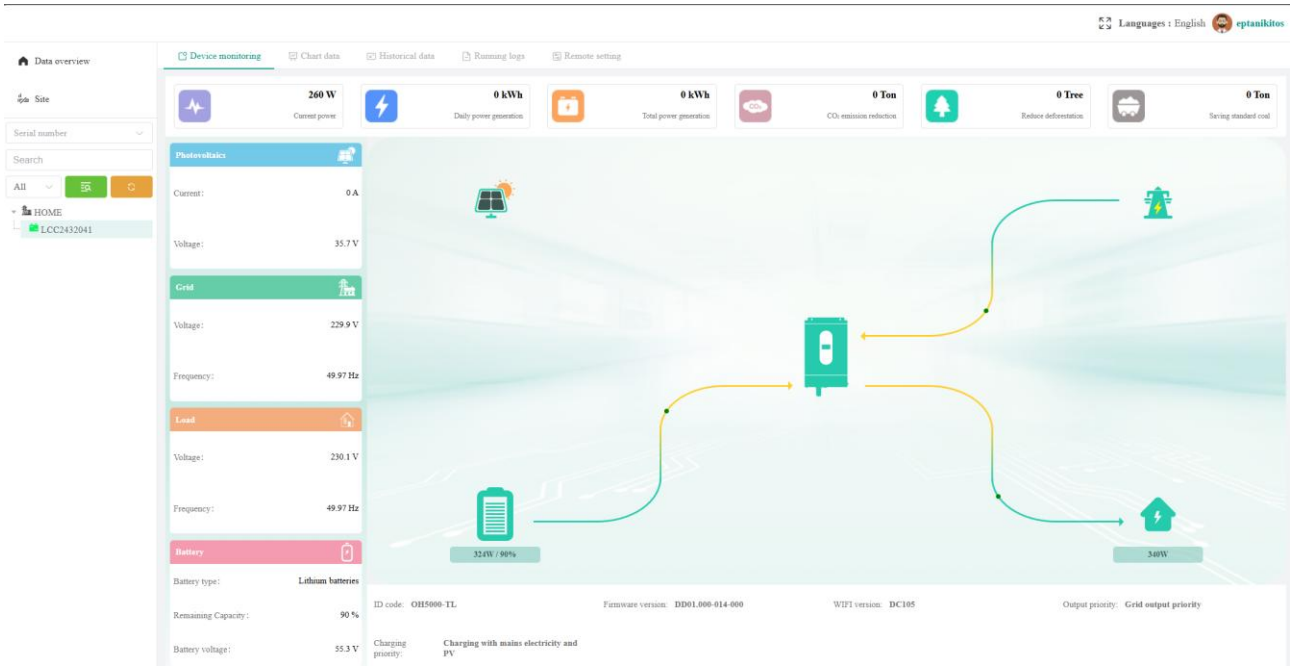


Рис. 2.2 Приклад типового вигляду веб-інтерфейсу зовнішнього моніторингового середовища mini-ems зі списком зареєстрованих об'єктів і станом інверторного обладнання

Штатний сценарій взаємодії побутового користувача із зовнішнім моніторинговим середовищем у його початковому вигляді, реконструйований за публічними матеріалами та підтверджений локальним кодом виявлення обладнання, охоплює: реєстрацію облікового запису користувача через веб-форму або мобільний застосунок, додавання об'єкта моніторингу (power station),

додавання та конфігурацію WiFi-модуля комунікації з обладнанням (collector або WiFi dongle), перегляд поточного стану інверторної установки, перегляд історії попереджень та помилок і виконання частини віддалених сервісних дій. Цей сценарій орієнтований на технічно грамотного користувача, що має одночасно великий екран і час на покрокове налаштування, та виконується одноразово або періодично – він не передбачає безперервного моніторингу із побутово зрозумілими сповіщеннями.

З погляду побутового сценарію, проаналізованого в підрозділі 1.2, такий штатний сервіс вирішує задачу інженерного огляду стану обладнання, але не закриває потребу в безперервному, побутово орієнтованому інформуванні. Користувач, який не перебуває біля стаціонарного комп'ютера або не має наміру щоразу заходити у мобільний застосунок виробника, не отримує миттєвої реакції на зміну стану зовнішньої мережі, не отримує заздалегідь сформульованої побутової відповіді на запитання про найближчу автономність роботи установки і не отримує безпосередньо в звичному месенджері спрощеного зведення стану. Більше того, представлення параметрів у штатному середовищі є технічним за термінологією, і безпосередньо непридатне для користувача, який не є кваліфікованим електриком або сервісним інженером.

Звідси випливає роль розробленої системи в цьому ланцюзі. У межах роботи бот не претендує на заміну офіційного сценарію встановлення та конфігурування обладнання, властивого зовнішньому моніторинговому середовищу виробника. Натомість він використовує mini-ems[8] як джерело даних, з якого отримуються поточні параметри роботи інверторної установки, та надбудовує над ними побутовий рівень подання. До цього рівня подання входять короткі та повні форми зведень стану, виражені побутово зрозумілою термінологією, активні сповіщення про зникнення та відновлення зовнішньої мережі, окремі тривоги про небезпечні відхилення напруги, прогноз очікуваної автономності резервного живлення, а також одноразова автентифікація з боку власника інвертора з подальшою прозорою для нього участю в моніторингу всього домогосподарства. Отже, з програмної точки зору mini-ems[8] є виключно зовнішнім середовищем автентифікації,

виявлення обладнання та постачання телеметрії, а вся побутова логіка системи реалізована саме в межах розробленої частини.

2.3. Порівняння підходів до побудови користувацького інтерфейсу

Окремою проєктною задачею є вибір форми реалізації користувацького інтерфейсу, через який кінцевий користувач отримує доступ до сформульованих у підрозділі 1.3 функціональних можливостей. Розгляд можливих варіантів доцільно вести стосовно тих самих побутових сценаріїв, що були виокремлені раніше: оперативне сповіщення про подію, безперервна доступність на мобільному пристрої, незначна початкова трудомісткість для користувача без технічної підготовки і сумісність із обмеженням мобільного середовища функціонування.

Першим варіантом є самостійний веб-інтерфейс, який передбачав би розгортання окремого веб-додатка з власним вебсервером, обліковими записами і власною логікою сесій. Перевагою такого підходу є висока гнучкість представлення і незалежність від платформи кінцевого користувача, але для розглядуваного класу побутових сценаріїв він тягне за собою низку істотних недоліків. По-перше, такий варіант вимагає окремої інфраструктури хостингу і доменного імені, що суперечить базовому проєктному обмеженню – функціонування системи на одиничному смартфоні в середовищі Termux[18]. По-друге, веб-інтерфейс за своєю природою є запитно-відповідальним: він не реалізує модель активного виштовхування подій до користувача, тому реакція на зникнення зовнішньої мережі або на небезпечну напругу можлива лише за умови, що користувач сам відкриє відповідну сторінку. По-третє, побутовому користувачу довелося б додатково запам'ятовувати окрему адресу, окремі облікові дані та підтримувати окремий інтерактивний канал, що знецінює саму ідею єдиного інтерфейсу для двох предметних доменів.

Другим варіантом є самостійний мобільний застосунок із власною інсталяцією на пристрої користувача. Цей варіант формально вирішує задачу активних сповіщень за рахунок системних push-сервісів, але одночасно різко

підвищує трудомісткість розробки та супроводу: для повноцінного функціонування він потребує розробки під одну або кілька мобільних платформ, проходження процедур публікації у відповідних магазинах застосунків, окремого циклу оновлень та забезпечення сумісності з рядом версій операційних систем.

Таблиця 2.2

Порівняння варіантів реалізації користувацького інтерфейсу

Критерій	Самостійний веб-інтерфейс	Мобільний застосунок	Бот месенджера Telegram
Необхідна інфраструктура	Власний хостинг, доменне ім'я, вебсервер, власні сесії	Жодної серверної частини не обов'язково, але потрібен цикл публікації і оновлень	Відсутня; використовується інфраструктура месенджера
Активні сповіщення	Неможливі без додаткових засобів	Через системні push-сервіси	Через штатний канал месенджера без додаткових налаштувань
Встановлення для користувача	Запам'ятовування адреси та облікових даних	Встановлення спеціалізованого застосунку	Перехід до бота за іменем, без встановлення
Кросплатформність	Залежно від браузера	Потрібна розробка під iOS та Android	Існує на смартфоні та настільному комп'ютері
Сумісність з Termux-обмеженням	Потребує серверної складової	Не усуває проблему окремої інсталяції	Повністю сумісний
Відповідність побутовому сценарію	Низька	Помірна; високий поріг входу	Висока

Не менш суттєвим є вплив на користувача: встановлення додаткового спеціалізованого побутового застосунку є для значної частини аудиторії помітною бар'єрною дією, особливо якщо застосунок не має широкої впізнаваності та

поширюється поза основними каналами розповсюдження. У контексті цільового сценарію, а це побутового моніторингу електропостачання та резервного живлення – окремий мобільний застосунок видається непропорційно ресурсозатратною формою для відносно компактної функціональності.

Третім варіантом є реалізація користувацького інтерфейсу у формі бота месенджера, зокрема в Telegram. Цей варіант принципово спирається на готову інфраструктуру месенджера, який уже встановлений у переважної частини цільової аудиторії, є кросплатформним, працює одночасно на смартфоні і на настільному комп'ютері та забезпечує штатний канал активного виштовхування повідомлень з боку серверної частини. Водночас на стороні розробки бот месенджера зводить кількість необхідних компонентів інтерфейсного шару до однієї підтримуваної реалізації, що використовує штатний публічний програмний інтерфейс месенджера. Для побутового користувача поріг входу зводиться до одного кроку – переходу до бота за відомим іменем та ніяка додаткова інсталяція або налаштування не потрібні.

З урахуванням усіх перелічених критеріїв обрано саме форму бота месенджера Telegram. Важливо при цьому зазначити, що такий вибір не зміщує тематичний акцент роботи: відповідно до обраного формулювання, розглядувана система описується як інформаційно-аналітична, а Telegram є саме інтерфейсним і доставковим шаром, через який реалізуються побутові сценарії взаємодії користувача з прикладною логікою моніторингу. Решта функціональних компонентів: фоновий моніторинг інвертора, фоновий моніторинг графіків відключень, локальне сховище історії, підсистема тривог, допоміжний інтелектуальний помічник та адміністративна підсистема – є самостійними і не залежать від конкретної реалізації інтерфейсу як форми. У межах роботи цей факт визначає логіку наступних розділів, в яких архітектура системи розглядається пошарово, а Telegram-шар лише як один з рівнів цієї архітектури.

2.4. Обґрунтування вибору технологічного стеку

З урахуванням обраної форми інтерфейсу та зафіксованого обмеження на мобільне середовище функціонування, технологічний стек сформовано в напрямку поєднання трьох якостей: достатньої виразної потужності для побудови описаної архітектури, мінімальної кількості нативних залежностей, придатних до встановлення в Termux[18] на 64-бітній архітектурі ARM, і збереження асинхронної моделі обробки повідомлень на рівні взаємодії з месенджером без втрати можливості використовувати усталені блокувальні бібліотеки для роботи з зовнішніми джерелами.

Базовою мовою програмування обрано Python[9]. Цей вибір зумовлено наявністю в Termux[18] готових пакетів інтерпретатора і налагодженого набору інструментарію для встановлення додаткових бібліотек, відсутністю необхідності в крос-компіляції під ARM для абсолютної більшості залежностей, властивих стека, і високою щільністю готових бібліотек у тих предметних областях, які безпосередньо потрібні для розглядуваної задачі: робота з програмним інтерфейсом месенджера, парсинг HTML, побудова HTTP-запитів, формування растрових зображень, робота з реляційним сховищем, симетричне шифрування облікових даних. Окрему перевагу Python[9] дає природна сумісність з асинхронною моделлю на основі asyncio, що відповідає сучасній моделі побудови ботів месенджера.

Для безпосередньої взаємодії з програмним інтерфейсом месенджера обрано бібліотеку aiogram[10] версії 3.x. Версія 3 цієї бібліотеки спирається на нову систему маршрутизаторів з явним декларативним описом обробників і підтримує зрілу інтеграцію з циклом подій asyncio. Це забезпечує природне поєднання обробки вхідних повідомлень користувачів з паралельною роботою фонових моніторингових циклів – обох ключових для розглядуваної системи: циклу опитування інверторного обладнання та циклу спостереження за станом графіків відключень.

Як локальне сховище даних обрано вбудовану СУБД SQLite[16]. Її ключовою перевагою для розглядуваного середовища є відсутність необхідності в окремому сервері бази даних, єдиний файловий артефакт даних, природна підтримка з боку штатних інструментів Termux[18] і простота резервного копіювання в межах операційних скриптів проєкту. Для рівня програмного відображення моделі даних обрано об'єктно-реляційний відобразник SQLAlchemy[15] версії 2.x, який забезпечує декларативний опис сутностей, явну роботу з сесіями, підтримку явних версіонованих міграцій схеми і одночасно зберігає можливість виконання прямих запитів за необхідності.

Для роботи з зовнішніми HTTP-джерелами використано поєднання двох усталених блокувальних бібліотек: requests[12] як HTTP-клієнт і BeautifulSoup[13] для парсингу HTML-структур сторінок неофіційного агрегатора графіків. Обидві бібліотеки є стандартом де-факто у відповідних задачах, мають широку документацію і встановлюються в Termux[18] без потреби в нативній компіляції. Їх блокувальна природа узгоджується з обраним архітектурним рішенням: відповідні виклики виконуються поза основним циклом подій через штатний механізм перенаправлення блокувального коду в окремий потік, що зберігає чуйність інтерфейсного шару при зверненнях до зовнішніх джерел.

Для формування растрових зображень графіків відключень, які надсилаються користувачу як відповідь у месенджері, обрано бібліотеку Pillow[14]. Її вибір зумовлено стабільною підтримкою в Termux[18], повнотою набору операцій, необхідних для формування таблично-композиційного зображення графіка, і відсутністю потреби у важких залежностях. Для симетричного шифрування паролів облікових записів користувачів зовнішнього моніторингового середовища обрано бібліотеку pyaes[17] у поєднанні з допоміжними алгоритмами розгортання ключа на основі PBKDF2/HMAC. Її перевагою у розглядуваному контексті є повністю чиста реалізація на Python[9], що знімає типові для криптографічних бібліотек складнощі з нативними збірками в обмеженому мобільному середовищі.

Таблиця 2.3

Технологічний стек системи

Компонент	Бібліотека / платформа	Версія	Функціональне призначення
Мова програмування	Python	3.x	Основна мова реалізації
Клієнт API месенджера	aiogram	3.26.0 (сім. 3.x)	Взаємодія з Telegram Bot API, маршрутизація подій
Реляційне сховище	SQLite	вбудована СУБД	Локальне постійне сховище даних
Об'єктно-реляційний відобразник	SQLAlchemy	2.x	Опис сутностей і сесій локального сховища
HTTP-клієнт	requests	стабільна	Звернення до зовнішніх HTTP-джерел
Парсер HTML	BeautifulSoup	4.x	Розбір сторінок зовнішнього агрегатора графіків
Генерація зображень	Pillow	стабільна	Побудова растрових зображень графіків
Симетричне шифрування	pyaes + PBKDF2/HMAC	стабільна	Шифрування облікових даних зовнішнього середовища
Зовнішня мовна модель	GitHub Models API	публічний API	Допоміжний інтелектуальний помічник

Як зовнішня служба генерації відповідей допоміжного інтелектуального помічника використовується програмний інтерфейс мовних моделей, доступних через інфраструктуру GitHub Models API[20], конфігурований через відповідний доступовий ключ у середовищі виконання. Ця інтеграція винесена в окремий сервісний модуль і не впливає на основні моніторингові цикли. Її вибір зумовлений наявністю готового HTTP-доступу до моделей без потреби в локальному запуску мовної моделі, що було б принципово несумісним з обмеженнями мобільного хост-середовища. Водночас, як зазначалося в загальній постановці, функціональність

інтелектуального помічника свідомо позиціонується у роботі як допоміжна, а не як ядро системи.

Важливо підкреслити, що обраний стек є послідовним продовженням обмеження на мобільне середовище функціонування. Жоден з його компонентів не вимагає окремого сервера баз даних, окремого брокера повідомлень, контейнерної середи виконання або зовнішнього процес-менеджера. Уся система може бути зведена до одного інтерпретатора Python[9], одного файлу бази даних SQLite[16], набору сценарних shell-скриптів для запуску і резервного копіювання та конфігураційного файлу секретів, що повністю узгоджується з обмеженням розгортання на одиничному смартфоні.

РОЗДІЛ 3. МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ TELEGRAM-БОТА

3.1. Вимоги до системи

Сформульовані у попередніх розділах функціональні очікування побутового користувача та обґрунтований у другому розділі технологічний контекст становлять змістову основу для формальної постановки вимог до інформаційно-аналітичної системи. У цьому підрозділі вимоги систематизовано за чотирма групами: функціональні, нефункціональні, безпекові та експлуатаційні. Такий поділ відповідає прийнятій у проєктуванні програмних систем практиці відокремлення зовнішньо спостережуваної поведінки від внутрішніх властивостей, які визначаються середовищем виконання та обмеженнями розгортання.

До функціональних вимог віднесено сукупність очікуваних реакцій системи на дії користувача та на події у зовнішніх джерелах. У межах розглядуваної системи функціональні вимоги структуровано за двома предметними доменами – інверторним моніторингом і моніторингом графіків погодинних відключень, а також за допоміжними можливостями. До інверторного домену належать такі вимоги: реєстрація власника інвертора з автоматичним виявленням ідентифікаторів обладнання у зовнішньому моніторинговому середовищі; приєднання іншого користувача до вже зареєстрованого фізичного інвертора без повторного введення облікових даних; формування короткого зведення про поточний стан інвертора без виконання прямого запиту до зовнішнього середовища у відповідь на кожне натискання кнопки; формування розгорнутого зведення з поясненнями полів; оперативне інформування про зникнення та відновлення зовнішньої мережі за фактом зміни стану; пороговий контроль напруги з окремими тривогами для перевищення значення 260В та для зниження нижче 170В; прогнозування орієнтовного часу автономної роботи на основі історії розряду батареї або на основі параметричної моделі обладнання. До домену графіків відключень належать вимоги: реєстрація персонального профілю користувача з прив'язкою до області та черги; запит поточного графіка на сьогоднішній і

завтрашній день у вигляді повідомлення з зображенням і підписом; автоматичне інформування про появу або зміну графіка за умови ввімкненої відповідної опції; коректна реакція на типові несправності зовнішнього джерела даних. До допоміжних функціональних вимог віднесено: обробку довільних текстових запитань користувача засобами зовнішньої мовної моделі з підстановкою поточного контексту моніторингу; адміністративний інтерфейс перегляду агрегованих відомостей про користувачів, накопичену телеметрію та активність системи.

Таблиця 3.1

Функціональні вимоги до системи

Домен	Вимога	Роль-ініціатор	Механізм активації
Інверторний моніторинг	Реєстрація власника з автоматичним виявленням ідентифікаторів обладнання	Власник інвертора	Користувацький запит
Інверторний моніторинг	Приєднання до вже зареєстрованого фізичного інвертора без повторного введення пароля	Приєднаний користувач	Користувацький запит
Інверторний моніторинг	Коротке зведення поточного стану	Власник / приєднаний користувач	Користувацький запит
Інверторний моніторинг	Розгорнуте зведення стану з поясненнями полів	Власник / приєднаний користувач	Користувацький запит
Інверторний моніторинг	Оперативне інформування про зникнення та відновлення зовнішньої мережі	Підписаний користувач	Фонова подія
Інверторний моніторинг	Порогові тривоги про напругу понад 260 В та нижче 170 В	Підписаний користувач	Фонова подія
Інверторний моніторинг	Прогноз орієнтовного часу автономної роботи	Підписаний користувач	Користувацький запит
Моніторинг графіків	Реєстрація профілю з вибором області та черги	Будь-який активний користувач	Користувацький запит

Продовження таблиці 3.1

Домен	Вимога	Роль-ініціатор	Механізм активації
Моніторинг графіків	Запит графіка на сьогодні та на завтра	Будь-який активний користувач	Користувацький запит
Моніторинг графіків	Автоматичне інформування про появу або зміну графіка	Підписаний користувач	Фонова подія
Моніторинг графіків	Коректна реакція на типові несправності зовнішнього джерела	Будь-який активний користувач	Автоматична
Допоміжні	Відповідь допоміжної мовної моделі з підстановкою поточного контексту	Будь-який активний користувач	Користувацький запит
Допоміжні	Адміністративний перегляд агрегованих відомостей	Адміністратор	Адміністративна команда

До нефункціональних вимог віднесено властивості, що характеризують якість виконання функціональних вимог у заданих умовах. Принципово важливою для розглядуваної системи є вимога незаблокованості інтерфейсу під час тривалих операцій із зовнішніми джерелами та локальною базою даних: оскільки взаємодія з користувачем відбувається через асинхронну подієву модель програмного інтерфейсу месенджера, тривалі мережеві запити, парсинг розмітки та операції з реляційним сховищем не повинні виконуватися безпосередньо у циклі обробки подій. Друга нефункціональна вимога стосується мінімізації навантаження на зовнішні джерела даних: одному фізичному інвертору, до якого підписані кілька користувачів, повинна відповідати одна ціль фонового опитування; одній парі «область – черга» в межах графіків відключень повинен відповідати один запит до зовнішнього сервісу за цикл моніторингу. Третя вимога фіксує очікування передбачуваної поведінки у разі тимчасової недоступності зовнішніх джерел: користувацький запит має повертати або найсвіжіший збережений варіант даних з відповідною позначкою, або змістовне повідомлення про природу проблеми, але не аварійну відмову. Четверта вимога стосується обмеженості споживання обчислювальних ресурсів і пам'яті, оскільки цільовим середовищем виконання є

одиничний смартфон, для якого недоступні характерні для серверних платформ запаси.

Таблиця 3.2

Нефункціональні, вимоги по безпеці та експлуатації

Категорія	Вимога	Мотивація
Нефункціональна	Незаблокованість асинхронного інтерфейсу під час тривалих операцій	Асинхронна модель Telegram Bot API
Нефункціональна	Одне фонове опитування на один фізичний інвертор / пару (область, черга)	Мінімізація навантаження на зовнішні джерела
Нефункціональна	Передбачувана поведінка у разі тимчасової недоступності зовнішнього джерела	Уникнення аварійних відмов
Нефункціональна	Обмежене споживання обчислювальних ресурсів і пам'яті	Розгортання на одиничному смартфоні
Безпекова	Шифроване зберігання облікових даних у постійному сховищі	Захист конфіденційних даних власника інвертора
Безпекова	Негайне видалення повідомлення з паролем після зчитування	Уникнення витоку пароля через історію листування
Безпекова	Розшифрування пароля лише в момент створення / відновлення сесії	Мінімізація часу перебування відкритого пароля в пам'яті
Безпекова	Захищена активація адміністративного інтерфейсу через службовий пароль	Обмеження доступу до агрегованих відомостей
Експлуатаційна	Запуск із командного рядка одним сценарієм	Зручність експлуатації у Termux
Експлуатаційна	Наглядовий перезапуск процесу	Тривала автономна робота
Експлуатаційна	Резервне копіювання локального сховища	Забезпечення збереження даних
Експлуатаційна	Перевірочний запуск для валідації конфігурації та схеми	Раннє виявлення проблем

До безпекових вимог віднесено властивості, що захищають облікові дані користувачів та конфіденційність взаємодії. Облікові дані власника інвертора, що використовуються для входу до зовнішнього моніторингового середовища, не можуть зберігатися у відкритому вигляді у постійній базі даних. Повідомлення з паролем, отримане через інтерфейс месенджера, повинне видалятися негайно після зчитування. Робочі сесії з зовнішнім середовищем повинні базуватися на кешованому токені, а не на повторному введенні пароля для кожного циклу опитування; розшифрування пароля допускається лише в момент відновлення або створення такої сесії. Адміністративний інтерфейс активується через окремий захищений сценарій із перевіркою службового пароля.

До експлуатаційних вимог віднесено властивості, що випливають із обраного середовища розгортання. Система повинна допускати запуск з командного рядка одним сценарієм, бути придатною до перезапуску засобами наглядового сценарію, а також підтримувати періодичне резервне копіювання локального сховища даних і створення архівних знімків стану. Окремою експлуатаційною вимогою є можливість попереднього перевірочного запуску, що валідує конфігурацію, доступність базових модулів і поточну версію схеми сховища. Усі ці вимоги є наслідком прийнятого у другому розділі обмеження мобільної платформи.

3.2. Моделі взаємодії користувачів із системою

Сформульовані у попередніх розділах функціональні очікування суттєво розрізняються за тим, який обсяг можливостей є доречним для конкретного користувача. Тому модель взаємодії з системою побудовано не як єдиний сценарій, а як набір ролей з диференційованим набором доступних дій. У межах роботи виокремлено п'ять ролей: гість, користувач без інвертора, власник інвертора, приєднаний користувач та адміністратор. Така деталізація дозволяє узгодити модель взаємодії з реальною структурою сутностей сховища даних, що розглядається у підрозділі 3.4, та з логікою спільного використання одного фізичного інвертора кількома користувачами в межах одного домогосподарства.

Гостем у роботі вважається користувач, який почав взаємодію з системою, але ще не пройшов жодного зі сценаріїв реєстрації. Для нього доступний лише обмежений сценарій ініціалізації, що демонструє три можливі гілки реєстрації та повідомляє про необхідність обрати одну з них перед використанням практичної функціональності. Обмеженість можливостей гостя є проєктним рішенням, що уникає неконтрольованого доступу до фонового моніторингу та до облікових даних інвертора.

Користувач без інвертора отримує доступ до повної функціональності, пов'язаної з графіками погодинних відключень. Йому доступні: формування персонального профілю з вибором області та черги; запит поточного графіка на сьогодні та завтра; перемикання режиму автоматичного інформування про появу або зміну графіка. Доступ до зведень про стан інвертора та до пов'язаних з ним сповіщень для цієї ролі не відкривається, оскільки такі дані є відсутніми за самим визначенням ролі. Окремо передбачена можливість пізнього додавання інвертора без втрати раніше сформованого профілю графіка.

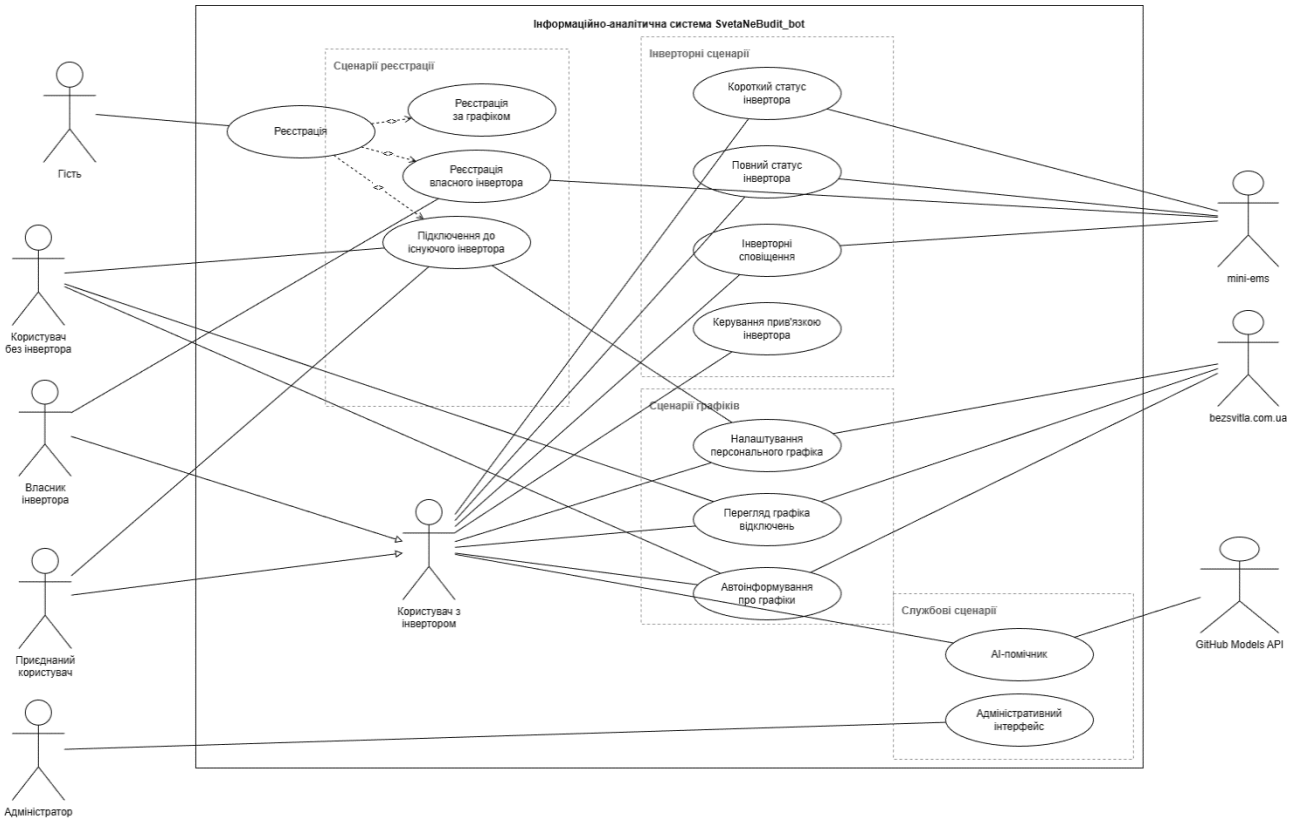


Рис. 3.1 Діаграма варіантів використання (Use Case) із ролей користувачів

Власник інвертора є роллю, у межах якої вперше встановлюється зв'язок системи з конкретним фізичним пристроєм. Сценарій реєстрації цієї ролі передбачає введення облікових даних до зовнішнього моніторингового середовища mini-ems[8] та подальше автоматичне виявлення ідентифікаторів об'єкта спостереження та конкретного інвертора. Власнику доступна повна функціональність системи: персональний профіль графіка; короткий і повний перегляд стану інвертора; перемикачі автоматичного інформування з обох доменів; керування прив'язкою інвертора, у тому числі заміна та відключення. Окремою особливістю ролі є відповідальність за фізичний пристрій: операція примусового відключення власника супроводжується відключенням усіх приєднаних до цього ж пристрою користувачів.

Приєднаний користувач становить логічну пару до власника інвертора в межах сценарію спільного домогосподарства. Він підключається до вже зареєстрованого у системі фізичного інвертора, надаючи лише вже відомий системі ідентифікатор облікового запису у зовнішньому моніторинговому середовищі. Облікові дані пароля при цьому повторно не вводяться, оскільки відповідна ціль фонового опитування вже існує. Приєднаному користувачу доступний той самий перелік зведень і сповіщень, що й власнику, але без права повного відключення фізичного пристрою. Така диференціація відповідає сформульованій у підрозділі 1.3 вимозі багатокористувацького використання та одночасно реалізує зазначене у підрозділі 3.1 нефункціональне обмеження одного фонового опитування на один фізичний пристрій.

Адміністратор активується окремою захищеною командою з перевіркою службового пароля. Йому доступні всі функції попередніх ролей у межах власного облікового запису, а також додатковий адміністративний інтерфейс із доступом до зведених відомостей: загальної кількості користувачів, кількості активних підписок з кожного домену, останнього зафіксованого стану мережі, кількості кешованих сесій із зовнішнім моніторинговим середовищем, поточної версії схеми сховища, кількостей сутностей у ключових таблицях та історії взаємодії з зовнішньою мовною моделлю. Адміністратор також є отримувачем технічних

попереджень про зміну структури сторінок зовнішнього агрегатора графіків. Адміністративний інтерфейс свідомо побудований як перегляд читабельних зведень, а не як панель керування налаштуваннями, що зменшує ризики випадкового пошкодження стану системи.

Таблиця 3.3

Ролі користувачів та їх доступні дії

Роль / можливість	Гість	Користувач без інвертора	Власник інвертора	Приєднаний користувач	Адміністратор
Ознайомлення з гілками реєстрації	Так	Так	Так	Так	Так
Персональний профіль графіка	Ні	Так	Так	Так	Так
Запит графіка на сьогодні / завтра	Ні	Так	Так	Так	Так
Автоінформування про зміну графіка	Ні	Так	Так	Так	Так
Коротке зведення стану інвертора	Ні	Ні	Так	Так	Так
Розгорнуте зведення стану інвертора	Ні	Ні	Так	Так	Так
Сповіщення про стан мережі та аномалії напруги	Ні	Ні	Так	Так	Так
Керування прив'язкою інвертора	Ні	Ні	Так (повне)	Частково	Так
Запитання до допоміжної мовної моделі	Ні	Так	Так	Так	Так
Адміністративна панель	Ні	Ні	Ні	Ні	Так

3.3. Архітектура програмної системи

Архітектуру розглядуваної системи побудовано як чотирьох рівневу модель, що відповідає природному розшаруванню між точкою взаємодії з користувачем,

прикладною логікою, локальним сховищем разом з оперативними буферами та зовнішніми джерелами даних. Такий поділ є наслідком функціональних та нефункціональних вимог, сформульованих у підрозділі 3.1, і відображає прийняте у другому розділі рішення розгорнути систему на одиничній мобільній платформі без застосування серверної інфраструктури.

Перший рівень. Рівень подання – складається з компонентів, безпосередньо звернених до користувача. До нього входять: ядро бота месенджера, побудоване на бібліотеці aiogram[10], з кастомним диспетчером, який доповнює стандартний обробник подій актор-орієнтованим протоколюванням; сукупність обробників, що реалізують команди, текстові гілки та реакцію на натискання кнопок; набори клавіатур, які формують різні розкладки головного меню для трьох сценаріїв (схема для користувача без інвертора, схема для користувача з інвертором, схема для адміністратора), а також inline-клавіатури вибору області, черги та керування прив'язкою інвертора. Цей рівень не виконує безпосередніх запитів до зовнішніх джерел: він делегує всю прикладну логіку наступному рівню.

Другий рівень. Прикладний (сервісний) – є центром реалізації прикладної логіки. До нього належать декілька самостійних сервісів. Сервіс сховища відповідає за всі операції з реляційною базою, у тому числі за початкове засіяння довідників ролей, перенесення спадкових даних, відновлення відсутніх зв'язків між користувачем і пристроєм, перенесення спадкових паролів у шифровану форму та політики ретенції. Сервіс моніторингу інвертора реалізує фонове опитування з визначеним інтервалом і виявлення подій зміни стану. Сервіс моніторингу графіків реалізує фонове опитування зовнішнього агрегатора з груповим запитом за областю. Сервіс прогнозу автономності формує орієнтовний час роботи від батареї за двоступеневою стратегією – на основі історії розряду або на основі параметричної моделі обладнання. Сервіс виявлення інвертора реалізує процедуру входу до зовнішнього моніторингового середовища та послідовного отримання ідентифікаторів об'єкта спостереження і конкретного пристрою. Сервіс взаємодії з мовною моделлю формує контекст із поточних показників і надсилає запит до зовнішнього програмного інтерфейсу мовних моделей. Сервіс шифрування

забезпечує симетричне шифрування облікових даних із використанням розгортання ключа на основі PBKDF2 / HMAC та режиму потокового шифрування. Допоміжний модуль форматування полів інвертора зберігає словник назв технічних показників для інтерпретації даних у структурований масив і використовується для побудови повного зведення стану.

Третій рівень. Рівень даних та оперативних буферів – поєднує постійне сховище у формі вбудованої реляційної бази SQLite[16] з керуванням через об'єктно-реляційний відобразник SQLAlchemy[15] та оперативні структури в пам'яті процесу. Постійне сховище містить ключові сутності, що детально розглядаються у підрозділі 3.4. Оперативні структури містять, зокрема: множину ідентифікаторів адміністраторів та активних підписників; буфер останнього отриманого знімка стану інвертора, окремий для кожного фізичного пристрою; буфери останніх знімків графіка, окремі для кожної комбінації області, черги та логічного дня; стан тривог за поточним інвертором; часові мітки заспокоювальних інтервалів для адміністративних попереджень та для повідомлень про відсутність графіка на наступний день; обмежений за довжиною журнал нещодавнього навантаження інвертора. Збереження частини стану в оперативній пам'яті є проектним наслідком вимоги уникати прямого запиту до зовнішнього джерела при кожній взаємодії з користувачем; постійне сховище при цьому залишається джерелом істини для відновлення стану після перезапуску процесу.

Четвертий рівень. Зовнішні джерела – об'єднує системи, які не входять до програмної реалізації роботи, але є необхідними для постачання даних. До цього рівня належать: зовнішнє моніторингове середовище виробника інверторного обладнання (mini-ems[8]); неофіційний сервіс агрегації та зручного подання графіків відключень (bezsvitla.com.ua[4]); програмний інтерфейс мовних моделей (GitHub Models API[20]); програмний інтерфейс месенджера (Telegram Bot API). Розмежування цього рівня з прикладним рівнем зафіксовано у другому розділі: вибір саме bezsvitla.com.ua[4] та саме mini-ems[8] як зовнішніх джерел даних є зовнішнім по відношенню до системи фактом і визначає характер інтеграції, але не входить до її внутрішньої структури.

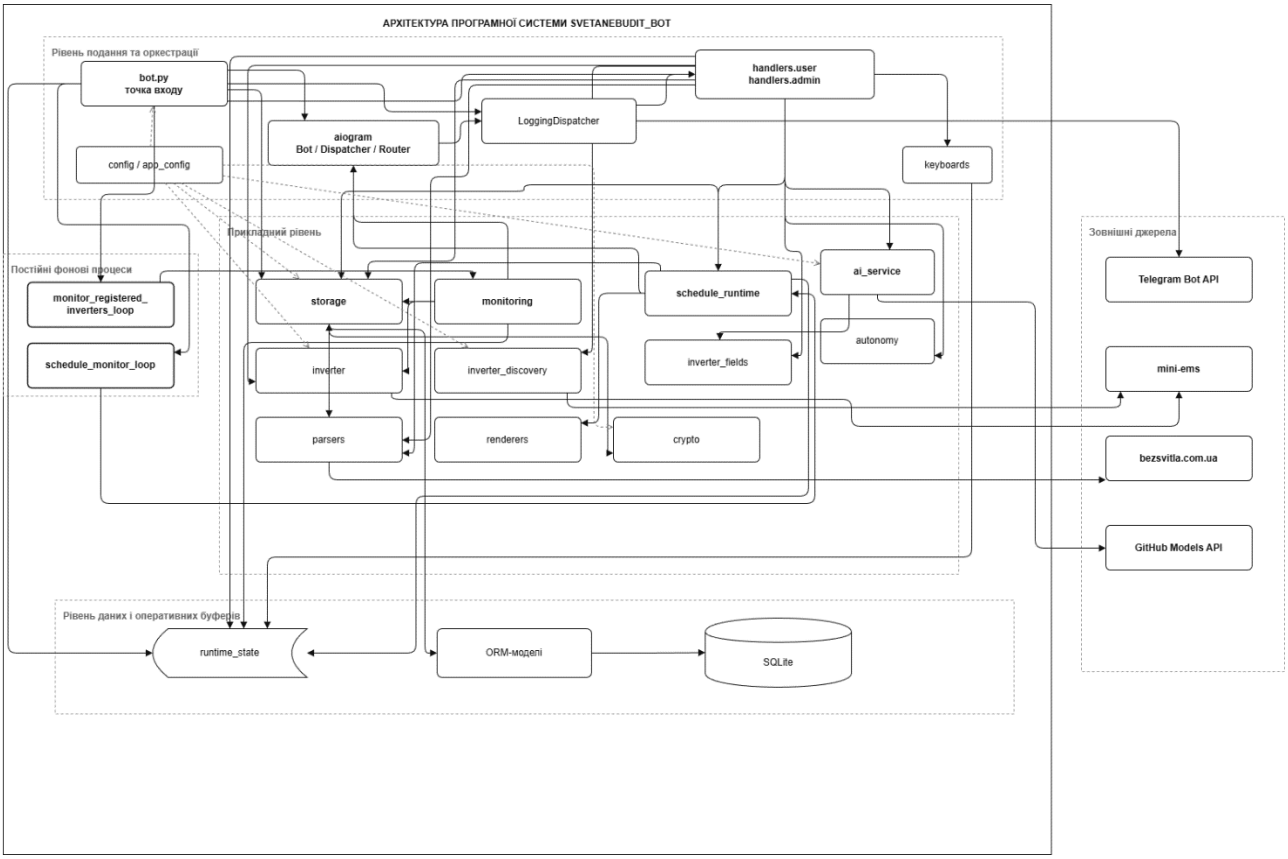


Рис. 3.2 Компонентна діаграма (Component) Архітектура програмної системи

Для коректного функціонування описаної архітектури в умовах подієво-асинхронного програмного інтерфейсу месенджера прийнято окреме архітектурне рішення щодо поведінки блокувальних операцій. Усі блокувальні дії – звернення до локальної бази даних, запити до зовнішніх HTTP-джерел, парсинг гіпертекстової розмітки та растрова генерація зображень – виконуються через делегування у пул системних потоків і не блокують основний цикл обробки подій. Це рішення дозволяє поєднати у межах одного процесу асинхронну взаємодію з месенджером і використання усталених синхронних бібліотек, що задовольняє відповідну нефункціональну вимогу з підрозділу 3.1.

Окремою архітектурною властивістю є наявність двох постійних фонових циклів, запущених при старті процесу та зупинених при його коректному завершенні. Перший цикл – моніторингу інвертора, виконується з інтервалом 5

секунд і опрацьовує групований перелік цілей моніторингу, отриманий зі сховища. Другий цикл – моніторингу графіків, виконується з інтервалом 600 секунд (десять хвилин) і опрацьовує перелік підписаних областей з обома денними горизонтами. Прийняте проектне рішення централізованого фонового моніторингу, замість одного незалежного звертання до сервісів на кожного користувача безпосередньо випливає з нефункціональної вимоги мінімізації навантаження на зовнішні джерела.

3.4. Проєктування сховища даних

Сховище даних розглядуваної системи побудовано на базі вбудованої реляційної СУБД SQLite[16] з керуванням через об'єктно-реляційний відобразник SQLAlchemy[15]. Цей вибір обґрунтовано у другому розділі і безпосередньо пов'язаний з обмеженнями мобільного середовища розгортання. Версіонування схеми реалізовано як послідовність явних кроків міграції, що утворюють лінійну історію розвитку моделі даних від першої версії до поточної. Така стратегія дає змогу безпечно еволюціонувати модель навіть в умовах ручного оновлення системи на одиничному пристрої, без необхідності в окремому інструменті керування міграціями.

Сутності сховища поділяються на чотири доменні групи: ідентифікація та доступ, інверторний домен, домен графіків відключень, допоміжна група для журналювання та системних метаданих. До групи ідентифікації та доступу належать сутності Role, User та UserState. Сутність Role зберігає реєстр доступних ролей користувачів, перерахованих у підрозділі 3.2. Сутність User містить ідентифікатор користувача в межах месенджера, посилання на роль та технічні поля журналу активності. Сутність UserState зберігає поточний стан користувача в межах кінцевого автомата діалогових сценаріїв, що дозволяє коректно відновлювати взаємодію після перезапуску процесу.

До інверторного домену належать сутності InverterAccount, InverterDevice, UserInverterLink та TelemetrySample. Сутність InverterAccount представляє

обліковий запис у зовнішньому моніторинговому середовищі і зберігає, зокрема, ідентифікатор користувача в зовнішньому середовищі, логін, шифровану форму пароля та посилання на користувача-власника в межах системи. Сутність InverterDevice представляє конкретний фізичний пристрій і ідентифікується композитною унікальною парою (power_id, inverter_id). Така унікальність є основним механізмом дедуплікації фізичного інвертора на рівні сховища даних: незалежно від того, скільки приєднаних користувачів спостерігають за пристроєм, у системі підтримується одна сутність InverterDevice. Сутність UserInverterLink опредметнює зв'язок «один користувач – один пристрій», що дає змогу одному фізичному пристрою бути пов'язаним з кількома користувачами через окремі активні зв'язки, але без породження повторних цілей моніторингу. Сутність TelemetrySample зберігає історичні відліки телеметрії конкретного пристрою з прив'язкою до часу.

Прийняте проєктне рішення розділити обліковий запис, фізичний пристрій і зв'язок користувача з пристроєм має кілька важливих наслідків. По-перше, воно дозволяє коректно описати ситуацію спільного домогосподарства, де один пристрій належить одному власнику, але доступний кільком приєднаним користувачам. По-друге, воно дозволяє реалізувати дедуплікацію фонового опитування на рівні групування цілей моніторингу: канонічний обліковий запис, що використовується для опитування конкретного пристрою, визначається через прив'язку до пристрою, а у разі її відсутності – через резервну логіку отримання облікового запису власника. По-третє, при операціях відключення приєднаного користувача або заміни пристрою історичний запис телеметрії видаляється, щоб дотримуватися правил безпеки та конфіденційності.

Сутність ScheduleSubscription зберігає персональний профіль користувача – обрану область і чергу, людиночитані найменування цих параметрів та позначку увімкненого автоматичного інформування. Сутність ScheduleSnapshot зберігає знімки графіків за комбінацією області, черги та дня, з полями джерельної URL-адреси, мітки часу оновлення, стиснутого подання графіка та контрольної суми збереженого подання.

Таблиця 3.4

Зведений перелік сутностей локального сховища

Сутність	Доменна група	Призначення	Ключові зв'язки
Role	Ідентифікація / доступ	Реєстр доступних ролей користувачів	Користувач -> Роль
User	Ідентифікація / доступ	Користувач у межах месенджера	Має роль; пов'язаний з InverterAccount / ScheduleSubscription
UserState	Ідентифікація / доступ	Поточний стан діалогових сценаріїв	Один до одного з User
InverterAccount	Інверторний	Обліковий запис у зовнішньому середовищі	Належить User; пов'язаний з InverterDevice
InverterDevice	Інверторний	Фізичний пристрій; унікальна пара (power_id, inverter_id)	Посилається на канонічний InverterAccount
UserInverterLink	Інверторний	Зв'язок «користувач-пристрій»	Багато-до-багатьох між User та InverterDevice
TelemetrySample	Інверторний	Історичні відліки телеметрії	Прив'язка до InverterDevice
ScheduleSubscription	Графіки	Персональний профіль графіка	Прив'язка до User
ScheduleSnapshot	Графіки	Знімок графіка за (область, черга, день)	Самостійна; індексована
AiRequestLog	Допоміжна	Журнал взаємодії з мовною моделлю	Прив'язка до User
AppSetting	Допоміжна	Службові налаштування «ключ-значення»	Самостійна

Окремою деталлю реалізації, яка має бути коректно описана в роботі, є те, що поле контрольної суми у поточному стані містить хеш від стиснутого текстового подання графіка, а не від вихідного гіпертекстового документа сторінки. Через цю особливість поле слід трактувати як ідентифікатор знімка для цілей виявлення змін, а не як буквальний хеш вихідного джерела.

Окремої уваги заслуговує проєктне рішення щодо безпеки облікових даних. Пароль до зовнішнього моніторингового середовища, який вводиться власником інвертора у процесі реєстрації, зберігається у шифрованому вигляді в окремому

полі сутності InverterAccount. Алгоритмічна основа цього рішення відповідає прийнятій безпековій вимозі з підрозділу 3.1: розгортання ключа здійснюється за допомогою PBKDF2 / HMAC-SHA256 з достатньою кількістю ітерацій, шифрування виконується симетричним алгоритмом у потоковому режимі, цілісність даних забезпечується кодом автентифікації повідомлення, цілісний тег зберігається разом із зашифрованим текстом. Прийнятий формат шифрованого подання має явний префікс версії, що дозволяє в майбутньому змінити схему шифрування без втрати сумісності з раніше збереженими даними. У моделі даних залишене застаріле текстове поле для пароля, але воно використовується виключно для одноразового перенесення спадкових даних і очищується активним кодом одразу після перенесення.

Окремою групою проєктних рішень є політики ретенції накопичуваних даних. Зберігання історичної телеметрії обмежене конфігурованим часовим вікном; зберігання знімків графіків відключень – конфігурованою кількістю днів; журнал взаємодії з мовною моделлю – як кількістю записів на користувача, так і максимальним віком запису. Такі обмеження відповідають експлуатаційній вимозі обмеженості споживання простору на мобільному пристрої. Додатково використано принцип запису на зміну: новий відлік телеметрії зберігається лише за умови суттєвої відмінності від попереднього (за порогами зміни напруги, відсотка заряду чи навантаження) або при переході між станами «онлайн» і «офлайн». Це зменшує кількість записів у періоди стабільної роботи без втрати інформації про зміни режиму.

3.5. Проєктування основних сценаріїв

У цьому підрозділі розглянуто проєктування п'яти ключових алгоритмічних сценаріїв, що визначають поведінку системи у відповідь на типові події: реєстрація власника інвертора, приєднання до вже існуючого фізичного інвертора, отримання короткого зведення стану, отримання графіка погодинних відключень та фонова розсилка тривог. Ці сценарії покривають критичну частину функціональних вимог

із підрозділу 3.1 і відображають взаємодію між шарами архітектури, описаної у підрозділі 3.3.

Сценарій реєстрації власника інвертора починається після того, як користувач на етапі вибору гілки реєстрації вказує відповідну роль. Обробник інтерфейсу запитує логін до зовнішнього моніторингового середовища, після чого окремим повідомленням – пароль.

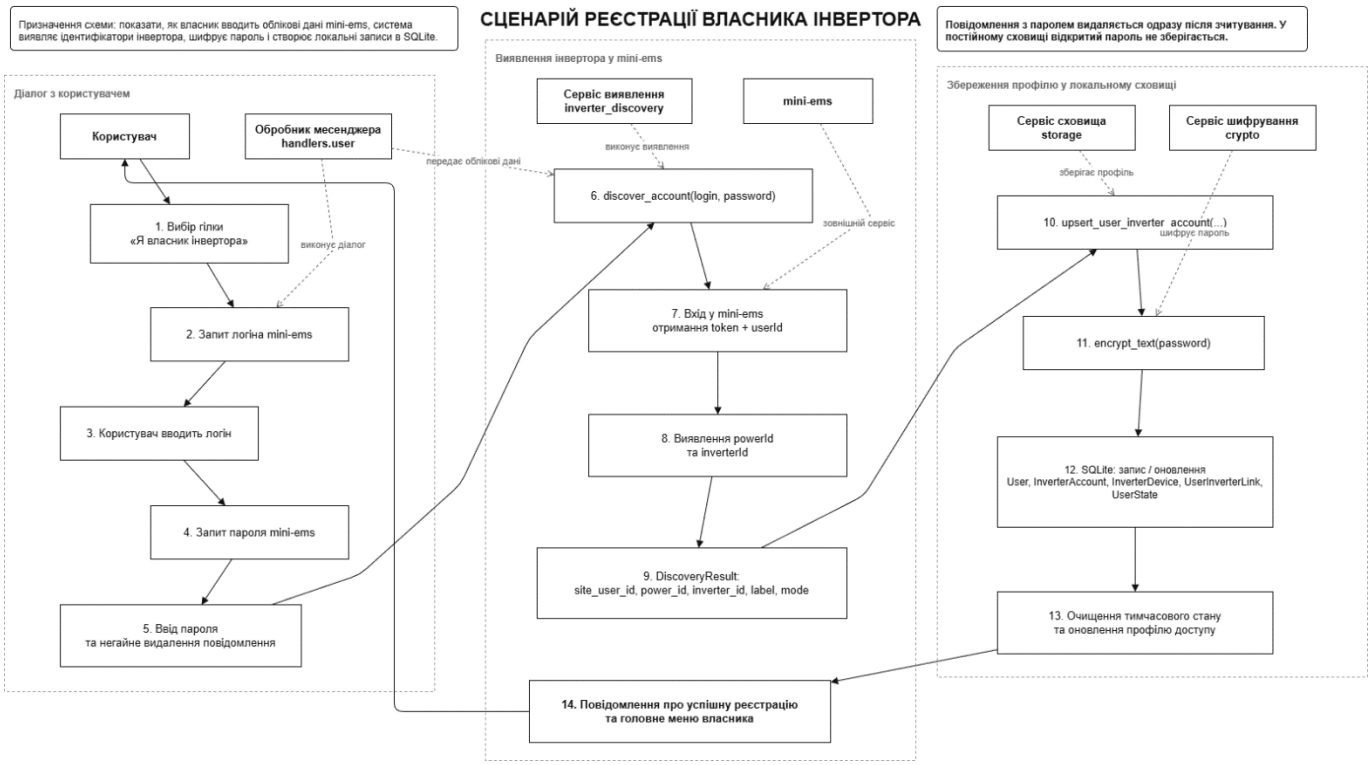


Рис. 3.3 Діаграма послідовності (Sequence) сценарію реєстрації власника

Повідомлення з паролем видаляється з історії листування одразу після зчитування, що відповідає безпековій вимозі підрозділу 3.1. Далі сервіс виявлення інвертора виконує послідовний ланцюг звернень до зовнішнього середовища: спочатку отримує ідентифікатор користувача за результатом входу, потім – ідентифікатор об'єкта спостереження за допомогою однієї з можливих варіантів запитів до відповідного програмного інтерфейсу, потім – ідентифікатор конкретного інвертора. Сервіс виявлення підтримує два режими взаємодії з зовнішнім середовищем – варіант для веб-інтерфейсу та варіант для мобільного застосунку, – що підвищує стійкість процедури до незначних відмінностей у

поведінці зовнішнього джерела даних. Отримані ідентифікатори передаються до сервісу сховища, який виконує операцію створення або оновлення сутностей InverterAccount, InverterDevice та UserInverterLink як єдиної послідовності з урахуванням композитної унікальності пристрою. Пароль зберігається у шифрованому вигляді. По завершенні сценарію користувачу повертається підтвердження успішної реєстрації та активується відповідна розкладка головного меню.

Сценарій приєднання до вже існуючого фізичного інвертора активується вибором відповідної гілки реєстрації. Користувач вводить лише логін облікового запису у зовнішньому моніторинговому середовищі, що вже відомий системі. Сервіс сховища знаходить за цим логіном активний обліковий запис і пов'язаний з ним активний пристрій, після чого створює нову сутність зв'язку UserInverterLink між користувачем і пристроєм або активує знову існуючий зв'язок, якщо такий був раніше. Принципово важливим є те, що в цьому сценарії жодна нова ціль фонового опитування не створюється: оскільки фоновий цикл моніторингу інвертора працює над згрупованим переліком цілей за фізичним пристроєм, додавання нового приєднаного користувача не спричиняє додаткових звернень до зовнішнього моніторингового середовища. Цей механізм безпосередньо реалізує нефункціональну вимогу підрозділу 3.1 щодо мінімізації навантаження на зовнішні джерела.

Сценарій отримання короткого зведення стану інвертора активується натисканням відповідної кнопки головного меню. Обробник інтерфейсу звертається до сервісу сховища за профілем доступу користувача, що містить, зокрема, ідентифікатор пов'язаного фізичного пристрою. За цим ідентифікатором у оперативному буфері останнього знімка стану виконується пошук відповідного запису. Якщо буфер містить актуальний знімок, він використовується як основне джерело відповіді. Якщо буфер на момент запиту порожній – наприклад, у разі першого звернення після перезапуску процесу до завершення першого циклу моніторингу – використовується резервний шлях у вигляді найсвіжішого збереженого відліку телеметрії з постійного сховища. Прийняте проектне рішення

цього сценарію свідомо виключає виконання прямого запиту до зовнішнього моніторингового середовища у відповідь на кожне натискання кнопки. Така поведінка обґрунтована двома міркуваннями: вона забезпечує однакову та швидку реакцію інтерфейсу на запит незалежно від поточної доступності зовнішнього джерела і одночасно реалізує нефункціональну вимогу мінімізації навантаження на зовнішнє джерело даних. Аналогічну логіку застосовано і до сценарію повного зведення, який також працює виключно над оперативним буфером.

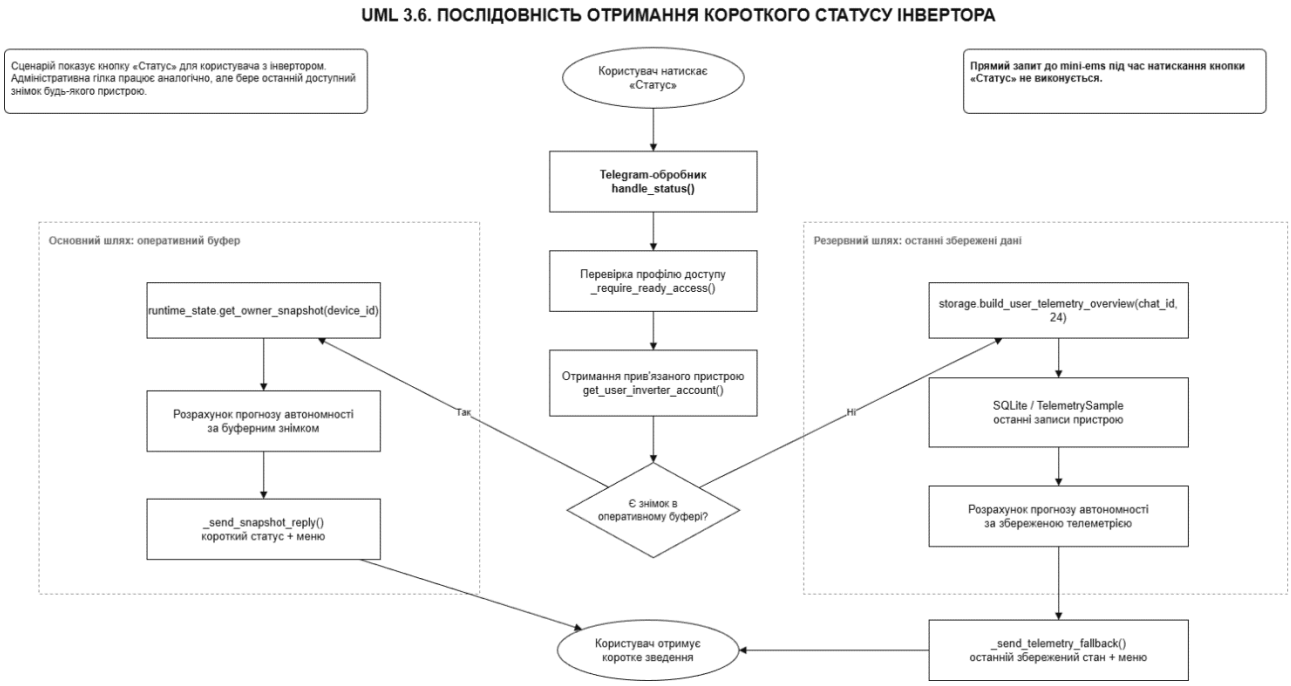


Рис. 3.4 Діаграма послідовності (Sequence) сценарію отримання статусу

Сценарій отримання графіка погодинних відключень побудовано як багатоступеневий ланцюг резервних шляхів подання, що окремо застосовується до кожного з двох денних горизонтів – поточного та наступного дня. Обробник інтерфейсу за персональним профілем користувача визначає область і чергу та послідовно виконує наступні спроби: пошук готового знімка у відповідному оперативному буфері; прямий запит до зовнішнього сервісу агрегації; пошук найсвіжішого збереженого знімка у постійному сховищі; для випадку «графік на сьогодні ще не опубліковано» – додаткова спроба отримати архівну сторінку за датою попереднього дня з відповідною позначкою у відповіді користувачу. Парсер

реалізує типізовану класифікацію невдач: тимчасова недоступність зовнішнього сервісу, зміна структури сторінки, відсутність черг для обраної області на поточний день, відсутність потрібної черги в межах коректно отриманої сторінки. Така класифікація дає змогу формувати дві різні реакції на ту саму подію: користувач бачить узагальнене повідомлення в простій формі без технічних подробиць, тоді як адміністратор окремим каналом інформується про подію типу «змінено структуру сторінки» з відповідним заспокійливим інтервалом, що уникає повторюваних повідомлень за тим самим приводом.

Принципово важливою властивістю розсилки тривог є її крайовий характер – повідомлення формується лише в момент переходу між станами і не повторюється при кожному циклі, що зафіксовано у відповідній експлуатаційній очікуваній властивості.

Таблиця 3.5

Ключові порогові значення і часові константи системи

Параметр	Значення	Сенс
Інтервал фонового циклу моніторингу інвертора	5 с	Періодичність опитування зовнішнього моніторингового середовища
Термін життя кешованої сесії	30 хв	Уникнення повторного входу на кожному циклі
Поріг розрізнення онлайн/офлайн	100 В	Фіксація факту наявності / зникнення зовнішньої мережі
Верхній поріг тривоги за напругою	260 В	Поріг активації сповіщення про небезпечно високу напругу
Нижній поріг тривоги за напругою	170 В	Поріг активації сповіщення про знижену напругу
Інтервал фонового циклу моніторингу графіків	600 с	Періодичність опитування зовнішнього агрегатора
Заспокійливий інтервал для повідомлень про відсутність графіка на завтра	12 год	Уникнення повторних сповіщень за тим самим приводом

Для сценарію зміни стану зовнішньої мережі це означає одне повідомлення в момент зникнення мережі та одне повідомлення в момент її відновлення, без проміжного періодичного спаму.

Поріг розрізнення станів «онлайн» і «офлайн» встановлено за значенням напруги вище 100В. Для сценарію відхилення напруги передбачено два пороги: верхній – більше 260В, нижній – менше 170В; повідомлення про вхід у відповідний аномальний стан надсилається одноразово в момент переходу і не дублюється при наступних циклах, доки стан не повернеться до нормального і потім знову не порушиться. Текст повідомлення про перевищення верхнього порогу містить практичну рекомендацію щодо безпечного відключення головного автоматичного вимикача на проміжок 30–40 хвилин з подальшою повторною перевіркою.

Сценарій фонового моніторингу графіків відключень побудований за аналогічним принципом крайового виявлення змін, але з істотно більшим інтервалом – 600 секунд (десять хвилин). Цикл опрацьовує групований за областями перелік цілей моніторингу: одна і та сама область, до якої підписано кількох користувачів з різними чергами, обробляється одним запитом до зовнішнього сервісу. Для кожної області запитуються обидва денні горизонти, після чого знімки порівнюються з раніше збереженими. У разі виявлення нової версії графіка або зміни поточної версії підписаним користувачам надсилається повідомлення з зображенням оновленого графіка та підписом, що містить мітку дня, дату, область, чергу та мітку часу оновлення.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА РОЗГОРТАННЯ СИСТЕМИ

4.1. Структура програмного проекту

Зафіксовані у попередньому розділі проєктні рішення реалізовано у вигляді одного застосунку мовою Python[9], побудованого за пакетною структурою з чітким розподілом відповідальності між пакетами. Структура репозиторію відповідає прийнятій чотирьох рівневій архітектурі: рівню подання відповідає пакет handlers/, рівню прикладної логіки – пакет services/, рівню даних – пакет db/; допоміжні задачі розбору гіпертекстових сторінок і візуалізації винесено у окремі пакети parsers/ та renderers/.

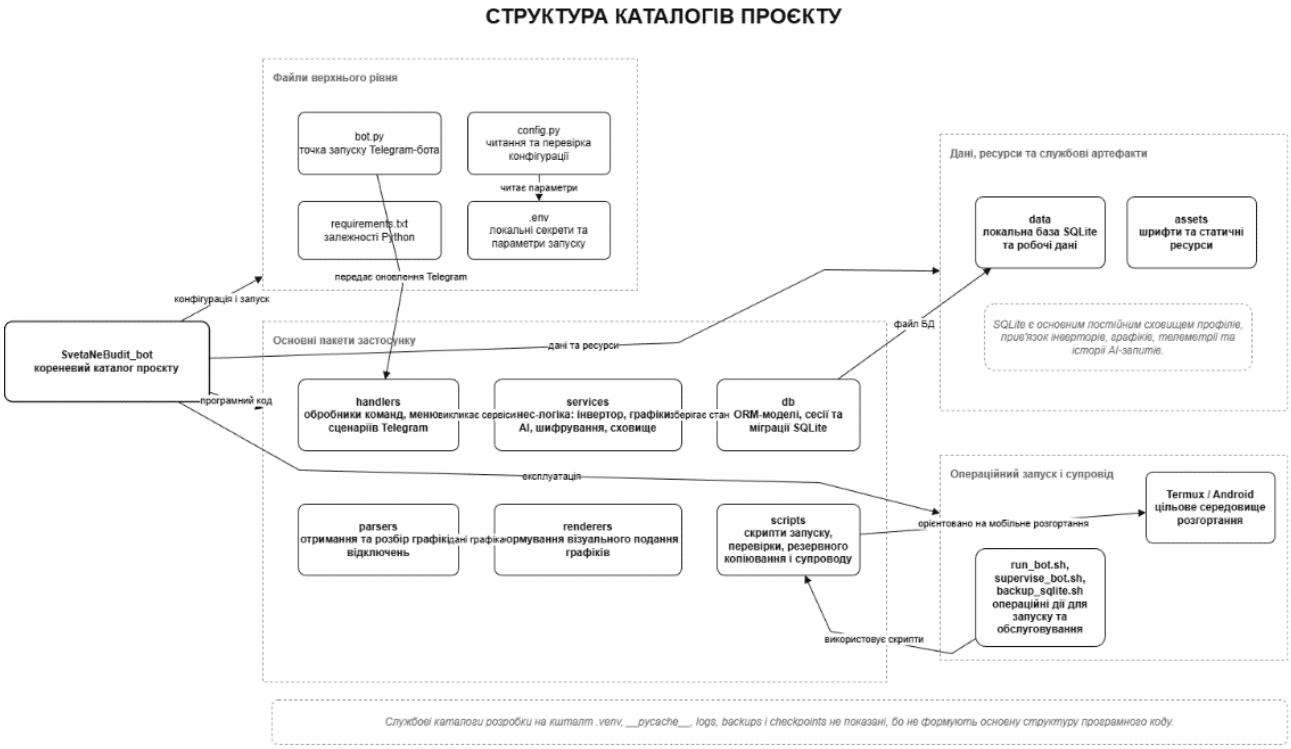


Рис. 4.1 Структура каталогів проекту з позначенням призначення основних пакетів

На верхньому рівні репозиторію розміщено точку входу застосунку bot.py і модуль конфігурації config.py, що відповідає за завантаження змінних середовища та валідацію обов'язкових налаштувань. Допоміжні сервісні сценарії для

експлуатації у мобільному середовищі згруповано у каталозі scripts/, локальне постійне сховище – у каталозі data/, шрифтові ресурси для візуалізації – у каталозі assets/. Така організація дає змогу розглядати кожний функціональний вузол як самостійну одиницю і знижує ризик небажаних зв'язків між рівнями архітектури.

Точка входу bot.py виконує функцію композиційного кореня застосунку. У ній послідовно ініціалізується сервіс сховища, завантажуються оперативні реєстри підписників і адміністраторів із постійного сховища, валідуються обов'язкові налаштування, створюється клієнт месенджера з розмічувальним режимом HTML, формується диспетчер подій з доданими маршрутизаторами для користувацького та адміністративного маршрутів, стартують два фонові цикли моніторингу і запускається цикл опитування подій месенджера. Завершення роботи процесу передбачає коректне скасування фонових задач і закриття мережевої сесії клієнта месенджера. Цей композиційний корінь свідомо не містить прикладної логіки: усе ділове наповнення винесене у пакет services/, що дозволяє розглядати bot.py як декларативний опис складу застосунку.

Пакет handlers/ містить три модулі: user.py з маршрутами користувацької взаємодії (реєстрація, перегляд стану, запит графіка, взаємодія з допоміжною мовною моделлю, керування прив'язкою інвертора), admin.py з маршрутами адміністративного інтерфейсу та keyboards.py з фабриками клавіатур відповіді і вбудованих клавіатур вибору регіону, черги та керування прив'язкою. Пакет services/ є найоб'ємнішим за кодовою базою і містить модулі, що реалізують прикладну логіку: storage.py (сервіс даних із розширеною діловою функціональністю), monitoring.py (фоновий цикл моніторингу інвертора та формування сповіщень), schedule_runtime.py (фоновий цикл моніторингу графіків і їх доставка), inverter.py (взаємодія із зовнішнім моніторинговим середовищем), inverter_discovery.py (ланцюг виявлення ідентифікаторів обладнання), autonomy.py (прогноз часу автономної роботи), ai_service.py (допоміжна взаємодія з зовнішньою мовною моделлю), inverter_fields.py (семантичне відображення технічних полів у просте подання), runtime_state.py (оперативні буфери у пам'яті процесу) та crypto.py (шифрування та перевірка цілісності облікових даних). Пакет

db/ містить модулі models.py, migrations.py та session.py, які реалізують опис сутностей, послідовність явних міграцій схеми та фабрику сесій реляційного з'єднання відповідно. Пакет parsers/ містить модуль schedule_parser.py з розбором сторінок зовнішнього агрегатора графіків, а пакет renderers/ – модуль schedule_image.py з побудовою зображення добового графіка. Допоміжний сервісний пакет scripts/ містить сценарії командної оболонки для перевірочного запуску, штатного запуску, наглядового перезапуску, резервного копіювання локального сховища та створення архівного знімка стану.

Таблиця 4.1

Відповідність основних модулів проекту їх призначенню

Пакет	Модуль	Рівень архітектури	Призначення
handlers/	user.py	Подання	Маршрути користувацької взаємодії
handlers/	admin.py	Подання	Маршрути адміністративного інтерфейсу
handlers/	keyboards.py	Подання	Фабрики клавіатур відповіді та вбудованих клавіатур
services/	storage.py	Прикладна логіка	Сервіс даних із розширеною діловою функціональністю
services/	monitoring.py	Прикладна логіка	Фоновий цикл моніторингу інвертора і формування сповіщень
services/	schedule_runtime.py	Прикладна логіка	Фоновий цикл моніторингу графіків і доставка
services/	inverter.py	Прикладна логіка	Взаємодія з зовнішнім моніторинговим середовищем
services/	inverter_discovery.py	Прикладна логіка	Ланцюг виявлення ідентифікаторів обладнання
services/	autonomy.py	Прикладна логіка	Прогноз часу автономної роботи
services/	ai_service.py	Прикладна логіка	Допоміжна взаємодія з мовною моделлю

Продовження таблиці 4.1

Пакет	Модуль	Рівень архітектури	Призначення
services/	inverter_fields.py	Прикладна логіка	Семантичне відображення полів інвертора
services/	runtime_state.py	Прикладна логіка	Оперативні буфери у пам'яті процесу
services/	crypto.py	Прикладна логіка	Шифрування та перевірка цілісності облікових даних
db/	models.py	Дані	Опис ORM-сутностей
db/	migrations.py	Дані	Послідовність явних міграцій схеми
db/	session.py	Дані	Фабрика сесій реляційного з'єднання
parsers/	schedule_parser.py	Допоміжний	Розбір сторінок зовнішнього агрегатора
renderers/	schedule_image.py	Допоміжний	Побудова зображення графіка
scripts/	*.sh	Експлуатація	Сценарії перевірочного запуску, запуску, нагляду, резервного копіювання, знімка

4.2. Реалізація Telegram-взаємодії та користувацьких сценаріїв

Рівень подання інформаційно-аналітичної системи реалізовано через маршрутизатори aiogram у пакеті handlers/. Користувацький маршрутизатор у модулі handlers/user.py обслуговує початкову взаємодію зі службовою командою /start, окремі команди /register, /schedule_profile, /status, /full_status, /schedule і узагальнений обробник довільного текстового повідомлення. Адміністративний маршрутизатор у модулі handlers/admin.py обслуговує команду /admin з позиційним параметром пароля, а також вбудовані клавіатури розділів адміністративної панелі. Принциповою особливістю рівня подання є узагальнений обробник текстового повідомлення: будь-який текст, що не збігається з жодним зі

службових маршрутів і надходить у момент, коли користувач не перебуває у стані кінцевого автомата діалогового сценарію реєстрації, інтерпретується як запитання до допоміжної мовної моделі, розглянутої у підрозділі 4.5.

Процес первинної реєстрації реалізовано як три самостійні гілки, що формально відповідають ролям, виокремленим у підрозділі 3.2: гілка реєстрації власника інвертора, гілка приєднання до вже зареєстрованого інвертора та гілка реєстрації для отримання графіків без подальшої прив'язки обладнання.

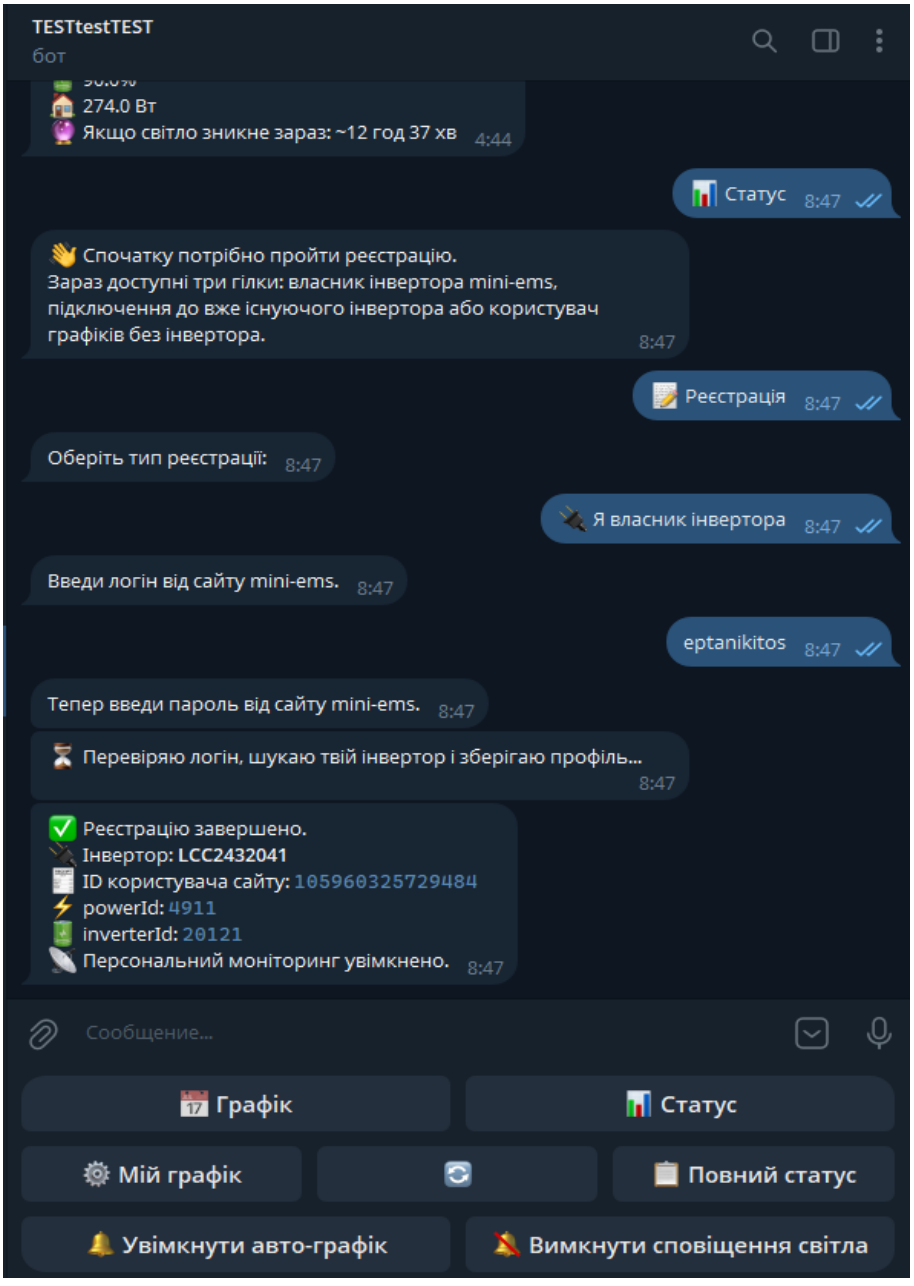


Рис. 4.2 Фрагмент діалогу реєстрації власника інвертора у клієнті месенджера

Вибір гілки здійснюється через клавіатуру відповіді з трьома кнопками з текстом « 🔌 Я власник інвертора», « 🔗 Підключитися до існуючого інвертора» та « 👤 Я без інвертора». Кожна гілка переводить діалог користувача у відповідну послідовність станів кінцевого автомата, реалізованого штатними засобами диспетчера подій.

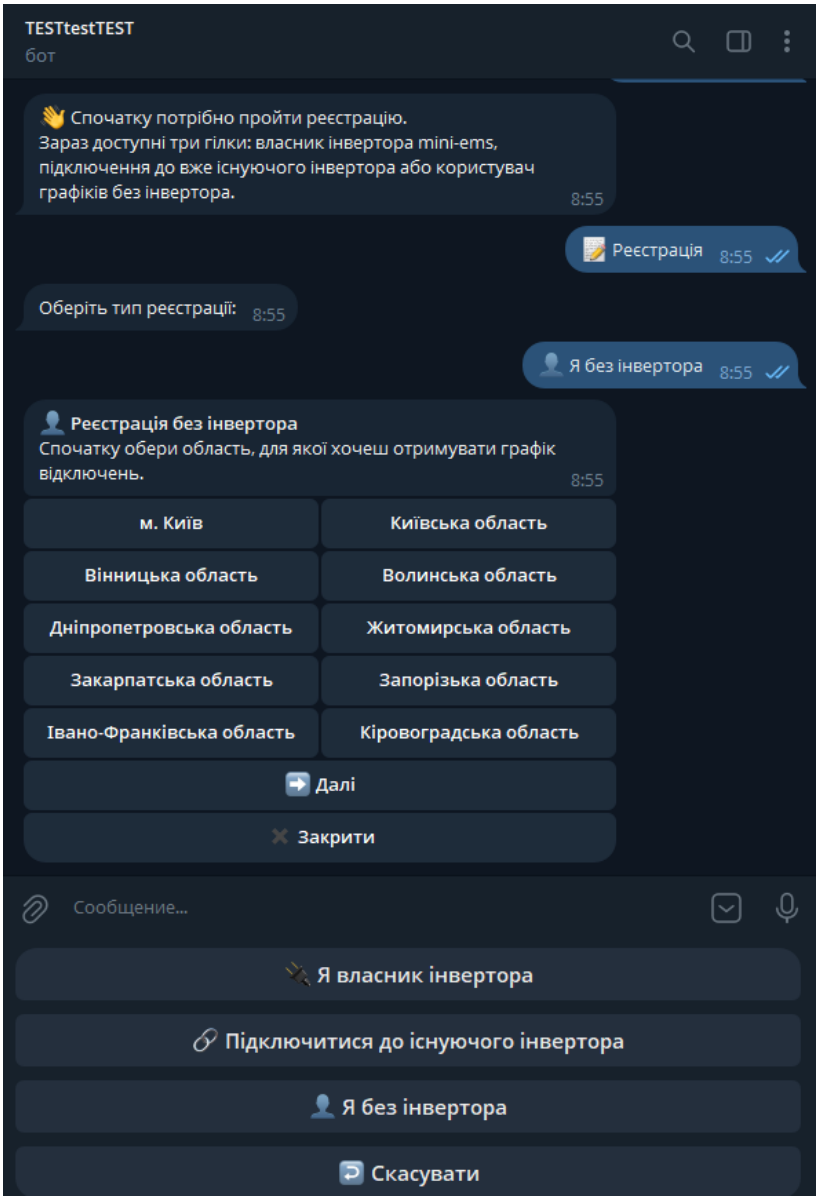


Рис. 4.3 Скріншот головного меню користувача без інвертора та запитом якої області вибрати графік

Для гілки власника інвертора послідовність станів охоплює: запит логіну для входу до зовнішнього моніторингового середовища виробника; запит пароля;

негайне видалення повідомлення з паролем із клієнта месенджера одразу після зчитування; виклик ланцюга виявлення ідентифікаторів обладнання (модуль services/inverter_discovery.py); збереження облікового запису, сутності фізичного пристрою та зв'язку користувача з пристроєм у локальному сховищі; формування головного меню власника інвертора. Для гілки приєднання послідовність коротша: користувач уводить лише логін, відомий системі; виявлення повторно не виконується; у локальному сховищі створюється або оновлюється зв'язок користувача з уже існуючим пристроєм, що було детально розглянуто у підрозділі 3.5. Для гілки користувача без інвертора запит логіну і пароля не здійснюється взагалі; користувач одразу переходить до діалогу формування персонального профілю графіка.

Головне меню, сформоване після завершення реєстрації, розрізняється за ролями і складається виключно з клавіатури відповіді. Для користувача без інвертора воно містить кнопку запиту графіка, кнопку налаштування персонального профілю графіка, кнопку пізнього додавання інвертора та перемикач автоматичного інформування про графік. Для власника інвертора або приєднаного користувача меню додатково містить кнопку короткого зведення стану, кнопку розгорнутого зведення, компактну кнопку керування прив'язкою інвертора, позначену символом «🔄», та окремий перемикач автоматичного інформування про стан інвертора.

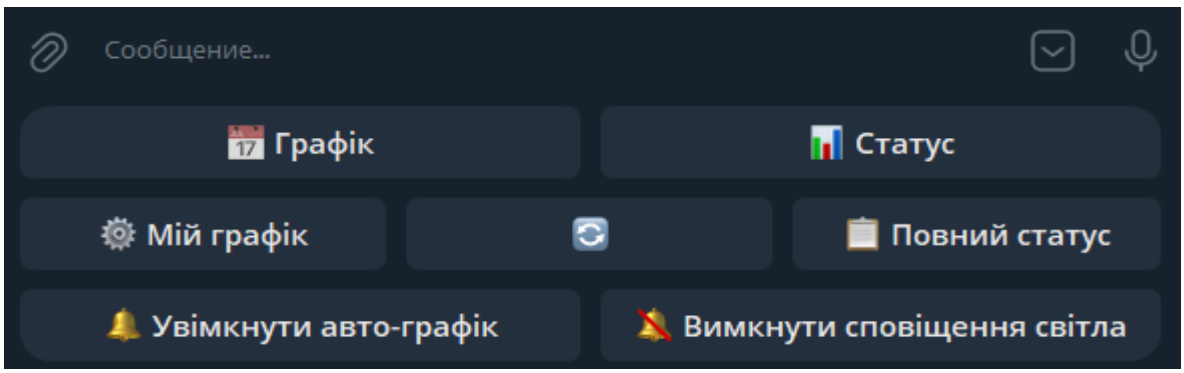


Рис. 4.4 Скріншот головного меню власника інвертора

Для адміністратора у меню додатково з'являється кнопка виклику адміністративної панелі.

Таблиця 4.2

Команди, кнопки та сценарії Telegram-інтерфейсу

Роль	Маршрут изатор	Маршрут / клавіатура	Опис реакції
Будь-який	user.py	/start	Вітання, перелік гілок реєстрації
Будь-який	user.py	/register	Повторний виклик обрання гілки реєстрації
Будь-який	user.py	Головне меню	Рольово-специфічна клавіатура відповіді
Без інвертора	user.py	Запит графіка	Відповідь зображенням графіка та підписом
Без інвертора	user.py	Налаштування профілю графіка	Вибір області, черги, перемикач авто-інформування
Без інвертора	user.py	Додати інвертор пізніше	Перехід у гілку реєстрації власника або приєднаного
Власник / приєднаний	user.py	/status	Коротке зведення з оперативного буфера / резерв зі сховища
Власник / приєднаний	user.py	/full_status	Розгорнуте зведення з словником назв полів
Власник / приєднаний	user.py	🔄 керування прив'язкою	Меню підключення / заміни / відключення пристрою
Власник / приєднаний	user.py	Перемикачі автоінформування	Увімкнення / вимкнення сповіщень
Власник / приєднаний	user.py	Довільний текст	Запитання до допоміжної мовної моделі
Адміністратор	admin.py	/admin <пароль>	Активація адміністративної панелі
Адміністратор	admin.py	Розділи панелі	Перегляд користувачів, статистики, телеметрії, історії AI, зведень

Побудова таких клавіатур інкапсульована у модулі handlers/keyboards.py, у якому реалізовано відповідні фабричні функції з аргументами, що враховують наявність прив'язки інвертора, наявність персонального профілю графіка та ознаку

адміністратора. Інтерфейс клавіатури вибору регіону та черги реалізовано через вбудовані клавіатури з форсованим переміщенням ключових для географічного контексту роботи позицій.

Користувацька мова інтерфейсу послідовно витримана українською: підписи кнопок, тексти повідомлень, повідомлення про помилки і службові повідомлення сформульовано українською мовою. Таке рішення узгоджене з цільовою аудиторією системи, сформульованою у першому розділі, і не передбачає перемикання мовних налаштувань у поточній версії. Окрему групу складають технічні повідомлення про зміну структури сторінки зовнішнього агрегатора графіків: такі повідомлення свідомо приховані від звичайних користувачів і надсилаються лише адміністраторам, що є проектним рішенням рівня подання, узгодженим з безпековою і експлуатаційною логікою системи.

4.3. Реалізація моніторингу інвертора і системи сповіщень

Фоновий цикл моніторингу інвертора реалізовано у модулі services/monitoring.py як асинхронну функцію monitor_registered_inverters_loop(bot), яка запускається з точки входу застосунку одразу після ініціалізації сховища та створення клієнта месенджера. Інтервал циклу становить 5 секунд. На кожному кроці циклу виконується фіксована послідовність дій: отримання з сервісу сховища згрупованого переліку цілей моніторингу; для кожної цілі – отримання нового знімка стану через модуль services/inverter.py; нормалізація отриманих даних у структуру InverterSnapshot; оновлення оперативного буфера знімків у пам'яті процесу; умовне збереження відліку телеметрії у постійне сховище; визначення переходу між станами зовнішньої мережі; визначення порогових відхилень напруги; формування та надсилання повідомлень підписаним на цю ціль користувачам. Перелік підписаних користувачів, яким надсилаються сповіщення, формується на постійному сховищі в межах того самого циклу, що дозволяє коректно реагувати на зміни складу підписників без перезапуску процесу.

Відсутність прямого запиту до зовнішнього сервісу у відповідь на користувацький запит короткого та розгорнутого зведення реалізовано за рахунок оперативного буфера знімків, що наповнюється виключно фоновим циклом. Обробники команд /status і /full_status у модулі handlers/user.py звертаються до функцій читання з буфера у модулі services/runtime_state.py. Для короткого зведення у разі, коли буфер ще не містить жодного знімка (наприклад, одразу після запуску процесу), реалізовано перехід до останнього запису телеметрії з постійного сховища, що виключає ситуацію порожньої відповіді, водночас не допускаючи синхронного прямого запиту до зовнішнього середовища всередині обробника. Для розгорнутого зведення використовується виключно оперативний буфер, а саме подання сформовано у модулі services/inverter_fields.py на основі словника семантичних відповідностей між технічними полями знімка та простішими описами. Оптимізація рівня подання полягає також у тому, що розгорнуте зведення не містить детальних переліків напруг вхідних кіл, які залишаються доступними у первинному буфері для внутрішнього використання і контексту допоміжної мовної моделі.

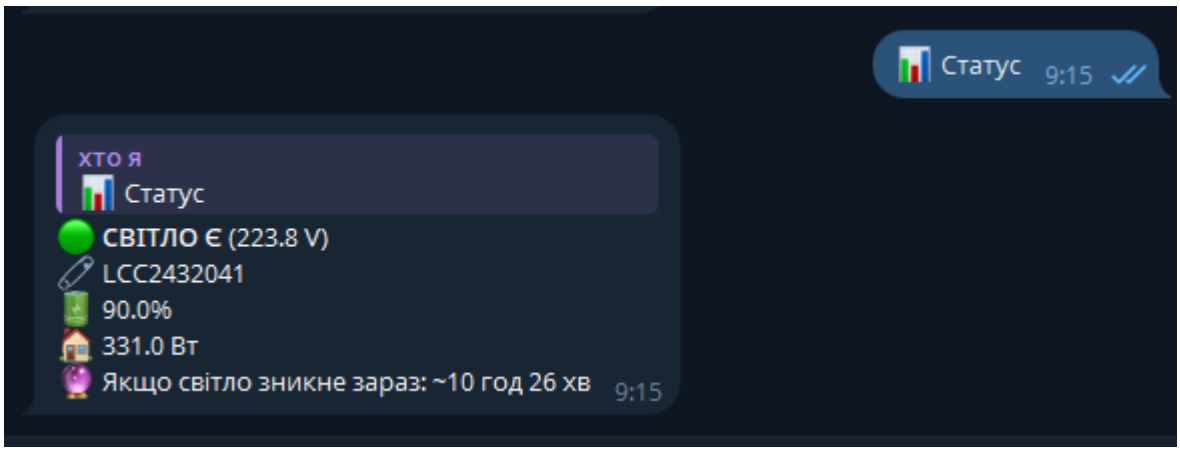


Рис. 4.5 Приклад короткого повідомлення `Статус` у клієнті месенджера

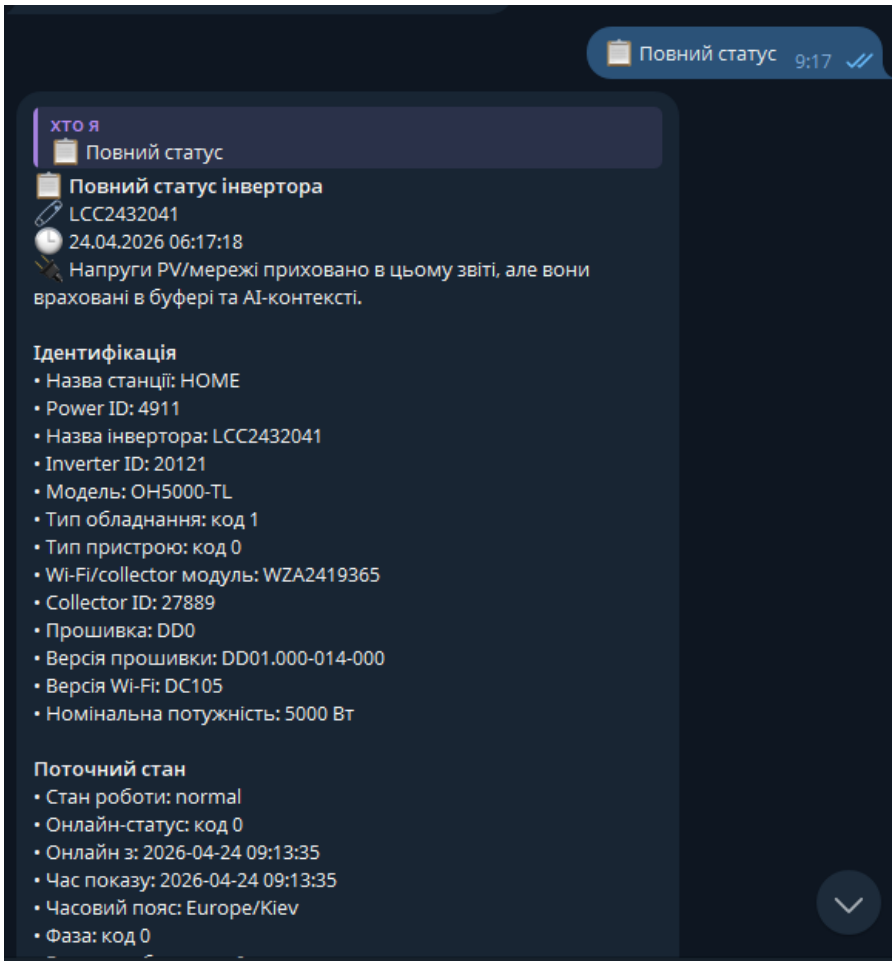


Рис. 4.6 Приклад розгорнутого повідомлення «Повний статус» у клієнті месенджера

Логіка сповіщень реалізована таким чином, що повідомлення користувачу формується лише в момент переходу до нового стану, а не на кожному кроці циклу. Для стану зовнішньої мережі поріг переходу до стану «мережа присутня» визначається порівнянням значення напруги вихідного кола з 100 В. Для порогового контролю напруги використовується високий поріг 260 В і низький поріг 170 В. Перехід до стану з перевищенням або заниженням напруги призводить до формування окремого сповіщення; повторне перебування у тому самому аномальному стані на наступних кроках циклу до формування додаткових сповіщень не призводить. Поточний стан за двома незалежними вимірами – стан зовнішньої мережі і стан порогового контролю напруги – зберігається окремо для кожного пристрою в оперативних буферах owner_inverter_statuses та

owner_voltage_alert_statuses модуля services/runtime_state.py. Сповіщення про перевищення високого порогу додатково містить стисле практичне формулювання про доцільність тимчасового відключення вхідного комутаційного пристрою на 30-40 хвилин з подальшою повторною перевіркою. Таке формулювання відповідає прикладному мотиватору, описаному у першому розділі, і свідомо адресоване побутовому користувачеві.

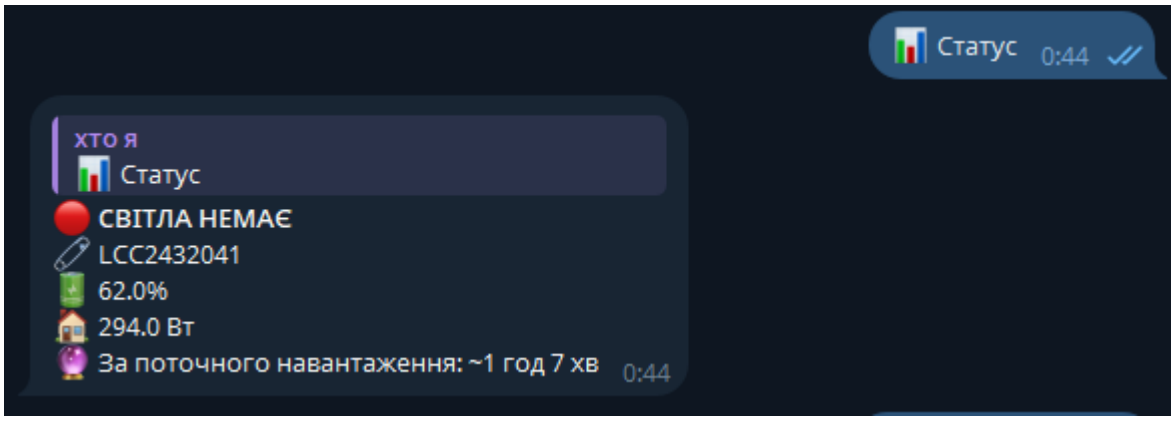


Рис. 4.7 Приклад повідомлення про відсутність зовнішньої мережі

Прогноз часу автономної роботи реалізовано у модулі services/autonomy.py і використовується у відповідях допоміжної мовної моделі на запитання, що явно стосуються автономності. У модулі передбачено двоступеневу стратегію: перший ступінь спирається на історію розряду батареї, отриману з накопиченої у постійному сховищі телеметрії; другий ступінь застосовується як резервний у випадку, коли обсяг наявної історії недостатній для статистичної оцінки, і будується на параметричній моделі обладнання з урахуванням раніше відомої номінальної ємності для родини пристрою. Прогноз використовує фіксовані константи: резервний відсоток заряду 15 %, коефіцієнт корисної дії перетворення 0,9, вікно усереднення навантаження 8 годин і нижню межу стабільного навантаження 30 Вт. Стратегія побудована таким чином, щоб не давати хибно оптимістичного прогнозу у випадку, коли спостережуване навантаження пульсує або історія розряду відсутня.

4.4. Реалізація підсистеми графіків відключень

Розбір гіпертекстових сторінок зовнішнього агрегатора графіків реалізовано у модулі parsers/schedule_parser.py на основі бібліотеки BeautifulSoup[13]. Модуль інкапсулює формування адрес сторінок регіону і черги, отримання переліку доступних регіонів із головної сторінки сервісу, розбір сторінки конкретного регіону у перелік карток черг, перетворення кожної картки у послідовність з 48 станів півгодинних інтервалів доби та витягнення мітки часу оновлення сторінки. Принциповою особливістю модуля є типізована обробка помилок: випадки, у яких сторінка повертається, але її зміст не відповідає очікуваній структурі, розрізняються за окремими різновидами помилок – недоступність сайту, зміна структури сторінки, відсутність черг для вказаного регіону на поточну дату і відсутність конкретної черги на коректно структурованій сторінці. Така класифікація дає змогу рівню прикладної логіки обрати різну реакцію для різних випадків: для звичайного користувача надсилається стримане повідомлення, для адміністратора – технічне повідомлення про зміну структури, що логічно відокремлює природу проблеми від форми її подання.

Послідовність резервних сценаріїв отримання графіка для користувацького запиту організовано у модулі services/schedule_runtime.py. Для кожного з двох денних горизонтів: поточного і наступного; застосовується фіксований порядок перевірки джерел: оперативний буфер поточного процесу; прямий запит до зовнішнього сервісу через модуль розбору; резервне читання збереженого знімка з постійного сховища; додатковий резервний сценарій з архівною сторінкою попередньої дати для випадку, коли графік на поточний день ще не опубліковано на головній сторінці регіону. Такий порядок дає змогу за одну ітерацію отримати найсвіжіше доступне подання графіка, а у разі неможливості отримати актуальне подання – повернути найсвіжіший історично доступний знімок з відповідною позначкою, що цей знімок походить з архіву. Окремою гілкою обробки є випадок відсутності графіка на наступний день: у цьому випадку користувачеві

надсилається стриманий текст «Не знайшов графік на завтра😔» без імітації порожнього зображення.

Представлення графіка у формі, зручній для подання у клієнті месенджера, реалізовано у модулі renderers/schedule_image.py на основі бібліотеки Pillow[14]. Модуль формує зображення з роздільною здатністю 800 × 880 пікселів у форматі PNG, на якому виведено дві таблиці по дванадцять годин, окрему легенду трьох станів (електроенергія присутня, відключення, невизначений стан) та, за потреби, виокремлене забарвлення поточного годинного інтервалу.

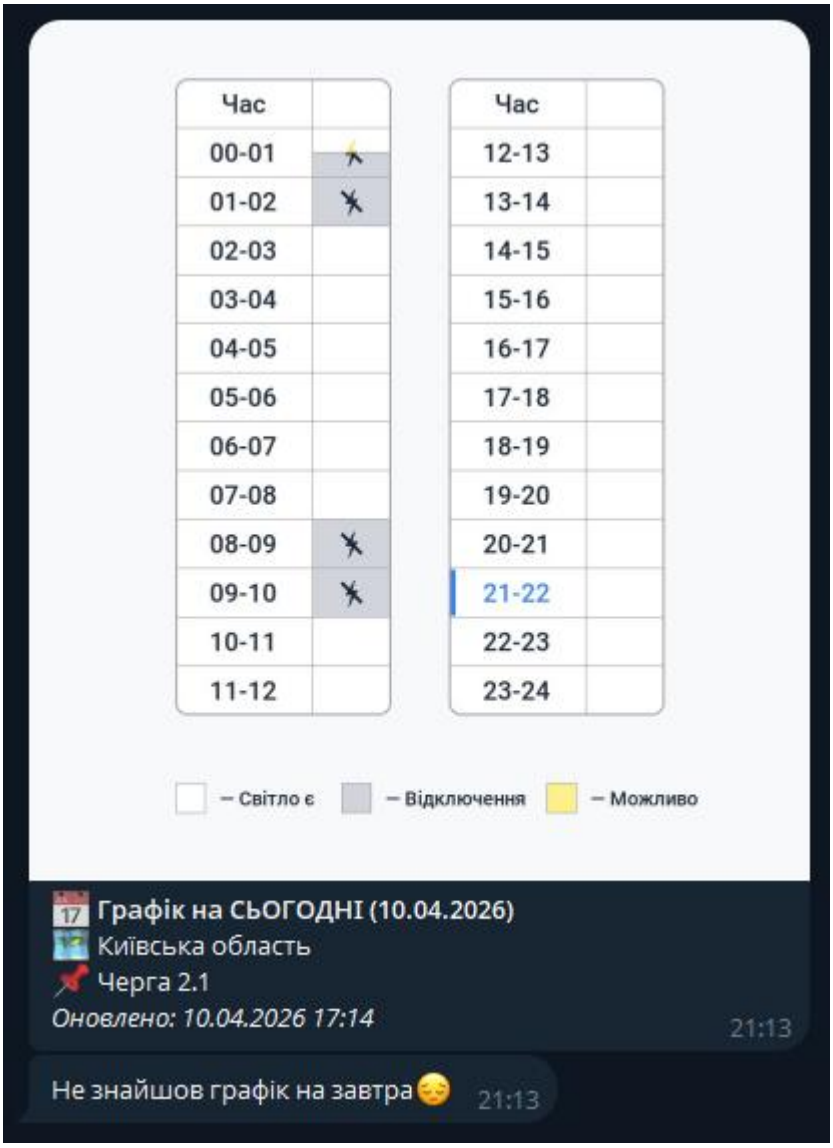


Рис. 4.8 Приклад сформованого зображення графіка відключень на поточну дату з двома таблицями по дванадцять годин

Для відображення написів перевага надається локальному шрифтовому ресурсу з каталогу assets/; у разі його відсутності передбачено аварійне завантаження шрифтового файлу з зовнішнього джерела, що є граничним сценарієм для типового розгортання, оскільки локальний шрифтовий ресурс присутній у репозиторії. Сформоване зображення надсилається користувачу засобами клієнта месенджера разом з підписом, що містить мітку дня, календарну дату, назву регіону, назву черги та мітку часу оновлення джерельної сторінки.

4.5. Реалізація адміністративних і AI-функцій

Адміністративний інтерфейс активується одноразовим уведенням команди /admin з позиційним параметром службового пароля. Маршрут обробника реалізовано у модулі handlers/admin.py. У разі успішної перевірки пароля ідентифікатор користувача додається до оперативного реєстру адміністраторів у пам'яті процесу і до постійного сховища; одночасно у головному меню користувача з'являється додаткова кнопка виклику адміністративної панелі. Сама панель реалізована як набір розділів, доступ до яких здійснюється через вбудовані клавіатури: перелік усіх зареєстрованих користувачів, агрегована статистика використання кнопок, зведення по накопиченій телеметрії, історія взаємодії з допоміжною мовною моделлю та загальне зведення стану системи. Загальне зведення містить кількість зареєстрованих користувачів, кількість активних підписників на інверторні сповіщення, кількість активних підписників на сповіщення про графіки, останній зафіксований стан зовнішньої мережі для спостережуваного пристрою, кількість кешованих робочих сесій із зовнішнім сервісом, поточну версію схеми сховища та підсумкові кількості сутностей інверторного домену. Ці відомості формуються через відповідні методи читання у сервісі сховища і не потребують окремих запитів до зовнішніх джерел.

Допоміжна мовна модель інтегрована у систему як окремий сервісний модуль services/ai_service.py. Принципово важливо, що ця підсистема розглядається у межах роботи як допоміжна інтерактивна функція над поточним

станом системи, а не як ядро розглядуваної інформаційно-аналітичної системи. Така позиція відображена як у проектних рішеннях, зафіксованих у попередньому розділі, так і в розподілі обсягу коду за пакетами: модуль взаємодії з мовною моделлю є одним з багатьох модулів пакета services/ і не займає у ньому центральної позиції. Активація цієї підсистеми відбувається через узагальнений обробник довільного текстового повідомлення: будь-який текст, що не збігається зі службовими маршрутами і надходить у момент відсутності активного діалогового сценарію реєстрації, інтерпретується як запитання користувача. Сервісний модуль формує системне повідомлення, що задає правила відповіді (стисла відповідь українською мовою, адаптація до тону користувача, прогноз часу автономної роботи лише у разі явного запиту), додає контекст поточного стану (час, середнє навантаження, історія навантаження, пояснений перелік полів інвертора, сирі дані інвертора, графіки на поточний і наступний день) і виконує запит до зовнішнього сервісу моделей із заданими параметрами повтору при тимчасових невдачах. Отримана відповідь надсилається користувачеві через клієнт месенджера.

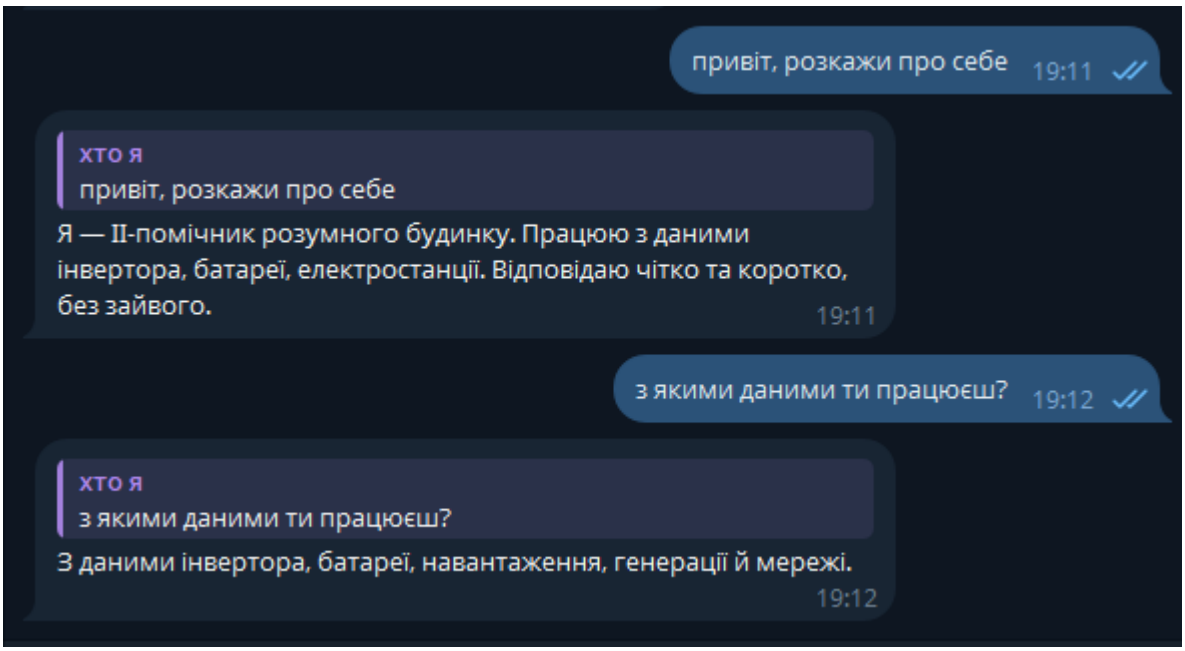


Рис. 4.9 Приклад взаємодії з мовною моделлю

Журналювання взаємодії з мовною моделлю реалізовано через сутність AiRequestLog. На кожну успішну відповідь у постійному сховищі зберігаються запитання, відповідь і часова мітка; відповідний лічильник дій у агрегованій статистиці інкрементується. Журнал підпадає під подвійну політику обмежень: кількісне обмеження історії на одного користувача та часове обмеження за віком запису. Така організація відповідає вимозі обмеженості споживання простору у мобільному середовищі і одночасно зберігає змістовний корпус взаємодії для адміністративного аналізу.

Окремим обмеженням поточної версії системи, яке слід коректно зафіксувати у контексті адміністративних і допоміжних функцій, є статус сутності GeneratedReport. Ця сутність присутня у моделі даних і супроводжується відповідними інфраструктурними гачками у сервісі сховища; разом з тим активний користувацький сценарій автоматичної генерації розгорнутих звітів засобами мовної моделі свідомо не входить до прикладної функціональності поточної версії. Обсяг виконаної роботи у частині допоміжної мовної моделі обмежений інтерактивною взаємодією з поточним станом системи; розширення до автоматичного генеративного звітування віднесено до можливих напрямів подальшого розвитку, що зафіксовано у висновках до розділу.

ВИСНОВКИ

У результаті виконаної дипломної роботи було створено інформаційно-аналітичну систему у формі Telegram-бота SvetaNeBudit_bot для побутового моніторингу електропостачання та резервного живлення, що об'єднує в єдиному мобільному інтерфейсі персоналізоване отримання графіків погодинних відключень, телеметрію IoT-сумісного інвертора та подійні сповіщення про стан зовнішньої мережі. Це дозволяє звести раніше розпорошені джерела інформації до одного зручного каналу взаємодії, забезпечити оперативне інформування користувача про зміну стану електроживлення і підвищити практичну керованість побутового сценарію використання резервного живлення.

У процесі роботи було виконано такі задачі:

1. Досліджено особливості побутового контролю електропостачання та резервного живлення в умовах України. Виявлено комплекс взаємопов'язаних задач, що одночасно постають перед типовим домогосподарством. У межах роботи здійснено детальний огляд поточного стану побутового електропостачання: зафіксовано нестабільність зовнішньої мережі, появу публічних графіків погодинних відключень як самостійного інформаційного об'єкта, ризики відхилень якості напруги у моменти присутності мережі та повсюдне поширення інверторних систем із акумуляторними батареями. Встановлено, що ці задачі не розв'язуються традиційними засобами побутової автоматики і потребують окремого інформаційно-аналітичного рішення, орієнтованого безпосередньо на побутового користувача.
2. Проаналізовано проблеми ручного моніторингу інвертора і графіків відключень та визначено функціональні очікування від майбутньої системи. Підтверджено, що кожне з існуючих джерел інформації у типовому сценарії потребує окремої свідомої дії користувача і за визначенням не забезпечує оперативного інформування. У межах аналізу встановлено, що штатне зовнішнє моніторингове середовище виробника інвертора вимагає окремої

автентифікації та самостійної інтерпретації технічних показників, офіційні ресурси публікації графіків побудовані навколо разового адресного сценарію, а неофіційні веб-агрегатори вимагають ручного звернення та залежать від стабільності зовнішньої розмітки. На основі зафіксованого прикладного інциденту з перевищенням напруги понад 310В, що не супроводжувався ніяким корисним попередженням з боку штатного середовища, сформульовано функціональні очікування від майбутньої системи: інформування про зміну стану мережі, інтегроване надання графіків відключень за групою користувача та підтримка багатокористувацького сценарію у межах одного домогосподарства.

3. Проаналізовано офіційні канали ДТЕК, сервіс bezsvitla.com.ua[4] та зовнішнє середовище mini-ems[8] як джерела даних для програмної системи. Виконано чітке розмежування трьох якісно відмінних типів зовнішніх джерел та визначено характер взаємодії з кожним із них. Офіційні ресурси ДТЕК опрацьовано як первинне джерело графіків погодинних відключень, bezsvitla.com.ua[5] як неофіційний агрегатор, що дозволяє перевідкрити ту саму інформацію у зручнішому для автоматичної обробки вигляді, а середовище mini-ems[8] – як зовнішню платформу виробника інверторного обладнання, на яку система спирається як на основне джерело телеметрії, не підміняючи штатного сервісного сценарію. Для кожного джерела описано характерні ризики та обґрунтовано логіку послідовних резервних сценаріїв отримання графіка, що забезпечує стійкість системи у разі часткової недоступності будь-якого з них.
4. Обґрунтовано вибір архітектурного підходу, користувацького інтерфейсу та технологічного стеку для реалізації системи. Прийнято рішення про реалізацію системи у формі Telegram-бота з розгортанням у мобільному середовищі без серверної інфраструктури. Як користувацький інтерфейс обрано месенджер Telegram, що забезпечує найнижчий поріг входу для побутового користувача та знімає потребу у встановленні окремого клієнтського застосунку. Обґрунтовано використання мови Python,

асинхронної бібліотеки aiogram[10] для роботи з Telegram Bot API,[11] вбудованої реляційної бази даних SQLite[16], об'єктно-реляційного відобразника SQLAlchemy[15] та допоміжних бібліотек requests, BeautifulSoup[13], Pillow[14] і pyaes[17]. Як цільове середовище розгортання обрано Termux на пристроях архітектури aarch64(arm), що дозволяє експлуатувати систему безпосередньо на смартфоні у режимі цілодобової роботи без окремого сервера. Обрано чотирьох рівневу архітектуру з розмежуванням рівня подання, прикладної логіки, оперативного стану в пам'яті процесу та постійного сховища.

5. Створено вимоги до системи, моделі взаємодії користувачів, архітектуру програмної системи, структуру сховища даних і основні сценарії її роботи. Розроблено комплексний проект майбутньої системи з дотриманням принципів модульності та розмежування відповідальності. Вимоги структуровано за функціональними, нефункціональними, безпековими та експлуатаційними групами. Розроблено модель ролей з чотирма категоріями учасників – власник інвертора, приєднаний користувач, користувач без інвертора та адміністратор, – що реалізує багатокористувацький доступ до одного фізичного пристрою у межах домогосподарства з розмежуванням прав. Структуру локального сховища спроектовано з дедуплікацією фізичного інвертора на рівні обмеження цілісності та зі зберіганням облікових даних у зашифрованому вигляді; описано основні алгоритмічні сценарії реєстрації, фонового опитування зовнішніх джерел, формування сповіщень з прикладно обґрунтованими порогами напруги 260В і 170В та побудови зображення графіка відключень засобами бібліотеки Pillow[14].
6. Реалізовано Telegram-бота, що об'єднує моніторинг інвертора, підсистему графіків відключень, систему сповіщень, адміністративні функції та допоміжні AI-можливості. Отримано завершену програмну систему, придатну до безпосередньої побутової експлуатації. Розроблено повноцінного Telegram-бота на базі aiogram[10] із підтримкою трьох гілок реєстрації користувачів, рольових варіантів головного меню, українського

інтерфейсу, буферизованого надання статусу без прямого зовнішнього запиту при кожному зверненні, фонового опитування інвертора з формуванням сповіщень у момент зміни стану зовнішньої мережі та у момент входу в аномальний діапазон напруги, а також фонового опитування графіків відключень з послідовністю резервних сценаріїв. Для адміністратора реалізовано окрему панель керування з можливістю контролю стану сервісів та обробки службових сповіщень про відхилення у роботі зовнішніх джерел. Додатково реалізовано допоміжний модуль взаємодії з великою мовною моделлю, що формує відповіді на основі поточного контексту моніторингу і розширює сценарії побутового користування системою.

Отриманий результат підтверджує практичну ефективність запропонованого рішення. Розроблена інформаційно-аналітична система дозволяє реалізувати централізований підхід до побутового моніторингу електропостачання та резервного живлення, орієнтований на реальні умови використання інверторних систем у домогосподарстві, а її розгортання у мобільному середовищі Termux без серверної інфраструктури робить рішення доступним для самостійного застосування побутовим користувачем.

ПЕРЕЛІК ПОСИЛАНЬ

1. Що потрібно знати про графіки відключень? – відповідає ДТЕК Київські електромережі [Електронний ресурс]. URL: https://www.dtek-kem.com.ua/ua/news/shcho-potribno-znati-pro-grafiki-vidklyuchen-vidpovidaye-dtek-kijivski-elektromerezhi (дата звернення: 23.04.2026).
2. Планові і аварійні відключення. ДТЕК Київські електромережі [Електронний ресурс]. URL: https://www.dtek-kem.com.ua/ua/shutdowns (дата звернення: 23.04.2026).
3. Аварійні відключення. ДТЕК Київські регіональні електромережі [Електронний ресурс]. URL: https://www.dtek-krem.com.ua/ua/shutdowns (дата звернення: 23.04.2026).
4. Про сервіс «Без світла» [Електронний ресурс]. URL: https://bezsvitla.com.ua/about (дата звернення: 23.04.2026).
5. Графік відключень світла у Києві. Без світла [Електронний ресурс]. URL: https://bezsvitla.com.ua/kyiv (дата звернення: 23.04.2026).
6. 5000TL_5Kw & 10000TL_10Kw Solar All-In-One ESS User Manual 1.01 [Електронний ресурс]. URL: https://www.delongtop.com/wp-content/uploads/2024/02/5000TL_5Kw-10000TL_10Kw-Solar-All-In-One-ESS-User-Manual-1.01.pdf (дата звернення: 23.04.2026).
7. 48V/51.2V 200Ah 10kWh All In One Energy Storage System With 10kW Inverter DL-LFP-51200 [Електронний ресурс]. URL: https://www.delongtop.com/product/51-2v-200ah-10kwh-all-in-one-energy-storage-system-dl-lfp-51200/ (дата звернення: 23.04.2026).
8. Mini EMS - Real Time Remote Energy Management System [Електронний ресурс]. URL: https://mini-ems.com/ (дата звернення: 23.04.2026).
9. Python 3 Documentation [Електронний ресурс]. URL: https://docs.python.org/3/ (дата звернення: 23.04.2026).
10. aiogram 3.27.0 documentation [Електронний ресурс]. URL: https://docs.aiogram.dev/en/v3.27.0/ (дата звернення: 23.04.2026).

11. Telegram Bot API [Електронний ресурс]. URL: https://core.telegram.org/bots/api (дата звернення: 23.04.2026).
12. Requests: HTTP for Humans [Електронний ресурс]. URL: https://requests.readthedocs.io/en/latest/ (дата звернення: 23.04.2026).
13. Beautiful Soup Documentation [Електронний ресурс]. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення: 23.04.2026).
14. Pillow Handbook [Електронний ресурс]. URL: https://pillow.readthedocs.io/en/stable/handbook/ (дата звернення: 23.04.2026).
15. SQLAlchemy 2.0 Documentation: Overview [Електронний ресурс]. URL: https://docs.sqlalchemy.org/en/20/intro.html (дата звернення: 23.04.2026).
16. About SQLite [Електронний ресурс]. URL: https://www.sqlite.org/about.html (дата звернення: 23.04.2026).
17. pyaes [Електронний ресурс]. URL: https://pypi.org/project/pyaes/ (дата звернення: 23.04.2026).
18. Termux App Docs [Електронний ресурс]. URL: https://termux.dev/en/docs/apps/termux/ (дата звернення: 23.04.2026).
19. termux/termux-app [Електронний ресурс]. URL: https://github.com/termux/termux-app (дата звернення: 23.04.2026).
20. GitHub Models [Електронний ресурс]. URL: https://docs.github.com/en/github-models (дата звернення: 23.04.2026).
21. Про затвердження Кодексу систем розподілу : постанова НКРЕКП від 14.03.2018 № 310 [Електронний ресурс]. URL: https://zakon.rada.gov.ua/go/v0310874-18 (дата звернення: 23.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Технологій цифрового розвитку

Інформаційно-аналітична система інтелектуального моніторингу та прогнозування енергозабезпечення на базі IoT-сумісних інверторів і графіків відключення світла

Виконав студент 4 курсу
Групи ТЦР-41
Бойченко Нікіта Олегович
Керівник роботи
Канд. техн. наук, доцент кафедри ТЦР Аронов Андрій Олексійович

Київ-2026

Мета, об'єкт та предмет дослідження

Мета роботи

Спрощення взаємодії побутового користувача з даними про електропостачання, резервне живлення, графіки відключень і параметри інвертора, за допомогою розробки аналітичної системи у вигляді Telegram-бота.

Об'єкт дослідження

Процес інформаційного супроводу побутового користувача під час контролю стану електропостачання, резервного живлення та графіків відключень електроенергії.

Предмет дослідження

Методи і програмні засоби побудови системи, що об'єднує телеметрію інвертора та публічні графіки відключень у єдиному побутовому інтерфейсі.

ⓘ Практичний фокус: користувач без окремих сайтів і застосунків бачить стан живлення, заряд батареї, прогноз автономності та персональний графік відключень.

Задачі кваліфікаційної роботи

1

Аналіз предметної області

Дослідити особливості побутового контролю електропостачання та резервного живлення в умовах України.

2

Виявлення проблем

Проаналізувати труднощі ручного моніторингу інвертора та графіків відключень; розглянути джерела даних ДТЕК, bezsvitla.com.ua, mini-ems.

3

Обґрунтування підходу

Обрати архітектурний підхід, інтерфейс і технологічний стек; спроєктувати ролі, сховище та ключові сценарії.

4

Реалізація системи

Розробити Telegram-бота з моніторингом інвертора, графіками відключень, сповіщеннями, адміністративними та AI-функціями.

Аналіз існуючих підходів

Підхід / Джерело	Сильна сторона	Обмеження	Висновок для системи
Офіційні канали ДТЕК	Офіційний характер інформації	Переважно адресний пошук і ручна перевірка	Потрібна персоналізована доставка графіка
bezsvitla.com.ua	Зручне подання графіків за областями і чергами	Залежність від доступності та HTML-структури сайту	Потрібні кешування і резервні сценарії
mini-ems	Доступ до телеметрії інвертора	Технічний інтерфейс і ручна автентифікація	Потрібне перетворення чисел у зрозумілий статус
Окремий web/app	Можливість власного інтерфейсу	Потреба у хостингу або інсталяції	Telegram-бот має нижчий поріг входу
Telegram-бот	Готовий канал сповіщень, кросплатформність	Обмеження формату інтерфейсу	**Обраний як основний канал взаємодії**

Вимоги до системи

Функціональні вимоги

- Реєстрація власника інвертора з автоматичним визначенням ідентифікаторів обладнання
- Підключення до вже зареєстрованого інвертора без повторного введення пароля
- Короткий і повний статус інвертора з поясненнями полів
- Сповіщення про зникнення/відновлення мережі та тривоги за напругою
- Персональний графік відключень і автоінформування про його зміни

Нефункціональні та безпекові вимоги

- Незаблокованість інтерфейсу під час тривалих HTTP-запитів
- Один фоновий опитувач на один фізичний інвертор
- Локальне сховище без окремого сервера БД та шифрування паролів
- Запуск у Termux на смартфоні

Програмні засоби реалізації

Python
Основна мова реалізації всієї системи

Requests + BeautifulSoup
Отримання і розбір зовнішніх HTML-джерел (bezsvitla.com.ua)

SQLite + SQLAlchemy ORM
Локальне постійне сховище; опис сутностей і керування сесіями

Aiogram 3.x
Взаємодія з Telegram Bot API та маршрутизація подій

Pillow
Генерація зображень персональних графіків відключень

pyaes + PBKDF2/HMAC
Шифрування облікових даних mini-ems

GitHub Models API
Допоміжний інтелектуальний модуль (LLM-інтеграція)

Termux
Мобільне середовище розгортання на Android-смартфоні

Діаграма варіантів використання системи

Ролі користувачів системи

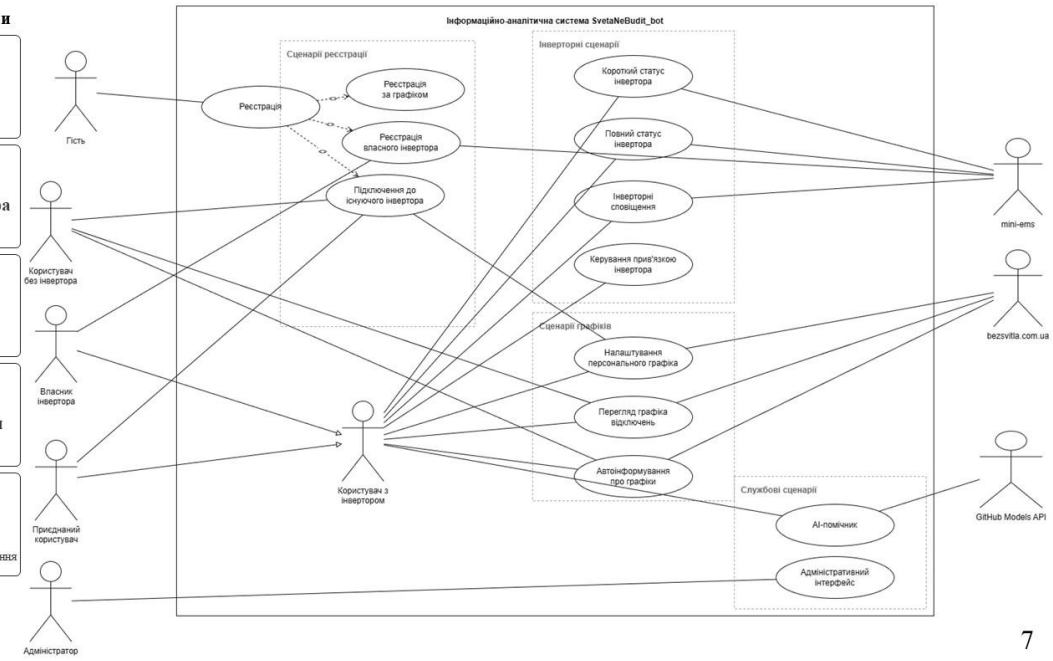
Гість
Реєстрація у системі

Користувач без інвертора
Перегляд графіків відключень

Власник інвертора
Повний моніторинг і сповіщення

Приєднаний користувач
Доступ до інвертора без реєстрації

Адміністратор
Адміністративний інтерфейс управління



Архітектура програмної системи

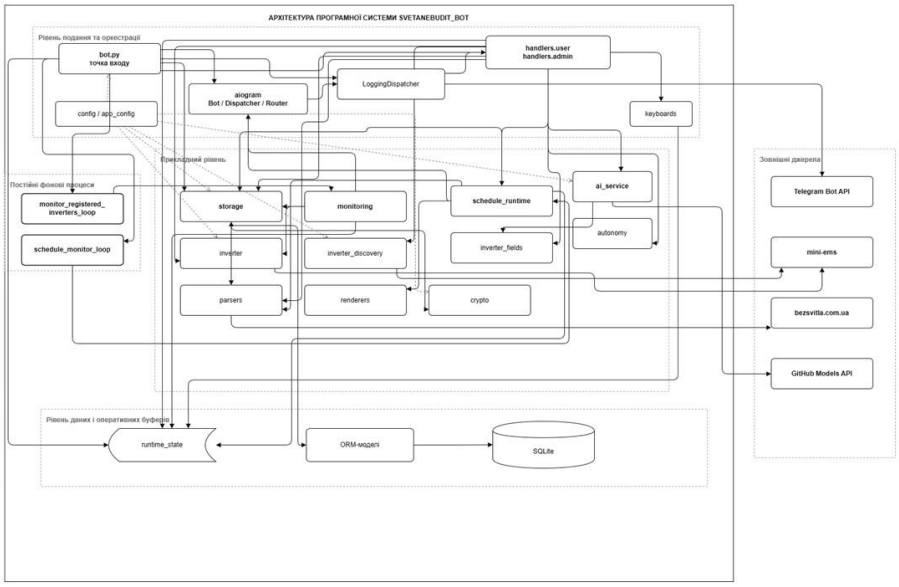
Ключові архітектурні рівні

Прикладний рівень
Сервіси інвертора, графіків, сховища, автономності та AI-модуль

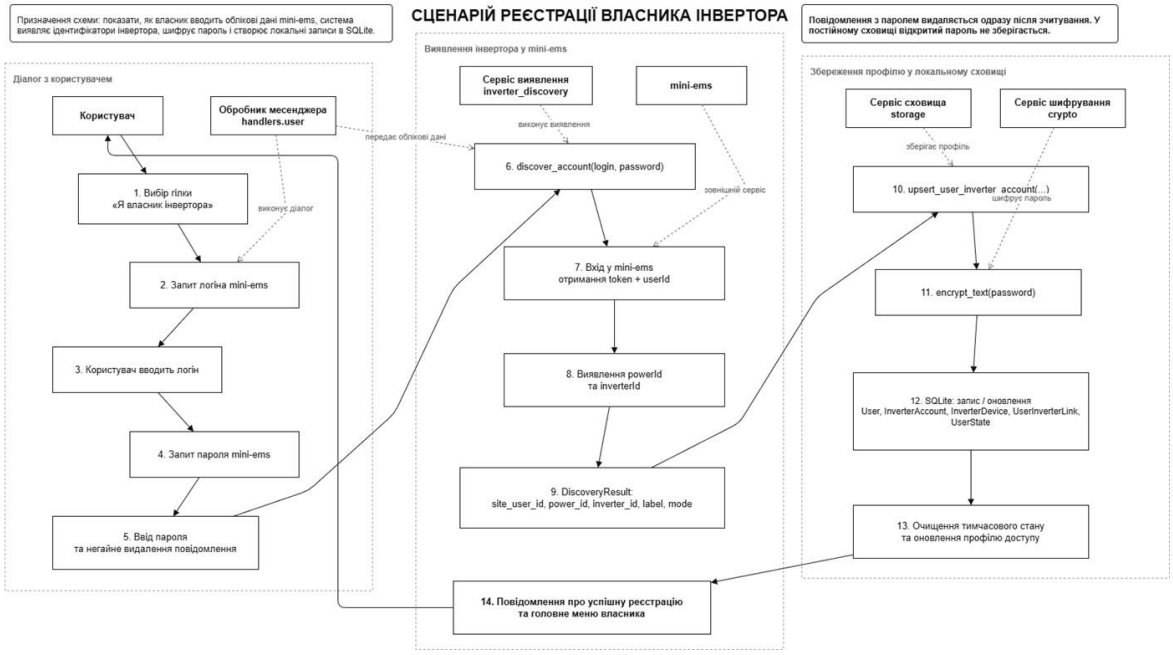
Рівень даних
SQLite, SQLAlchemy ORM, оперативні буфери runtime_state

Рівень подання
Telegram-обробники, клавіатури, диспетчер подій Aiogram

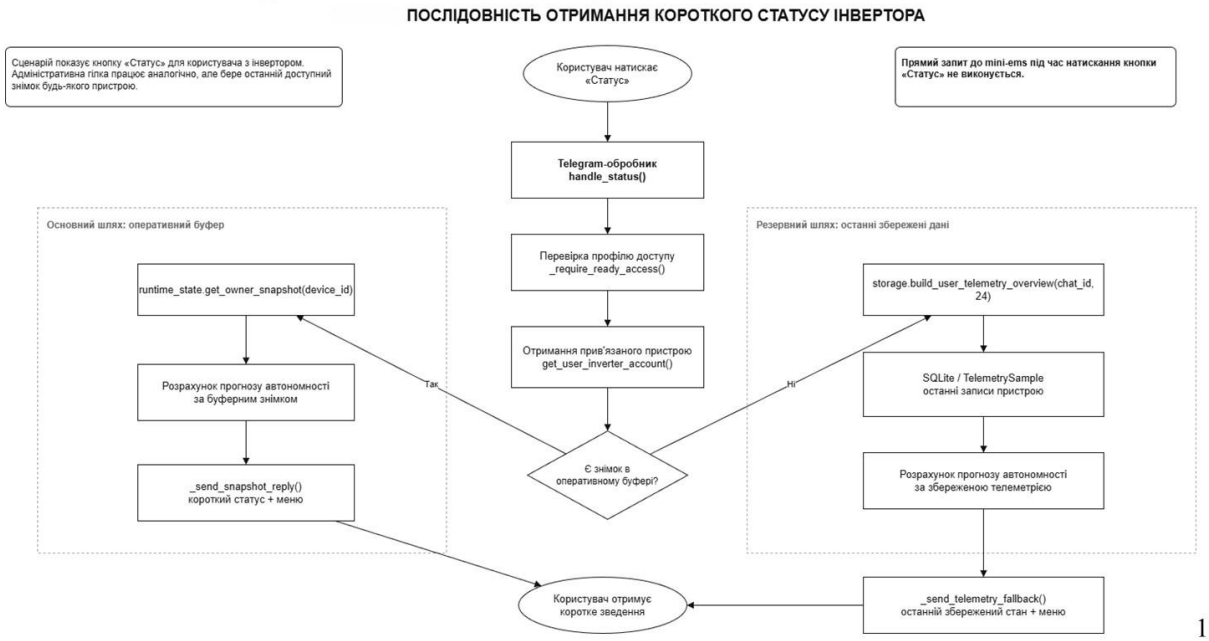
Зовнішні джерела
mini-ems, bezsvitla.com.ua, GitHub Models API



Основні алгоритмічні сценарії



Основні алгоритмічні сценарії



ТЕСТУВАННЯ

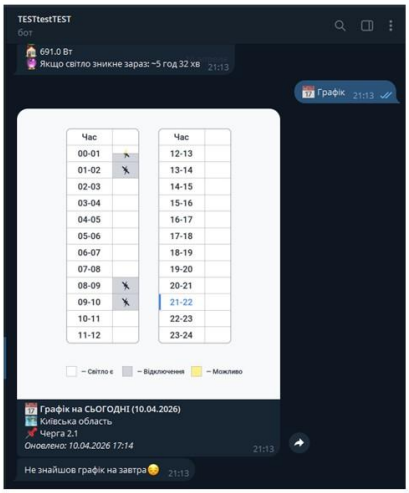
Для швидкої діагностики стану системи реалізовано bash-скрипт самоперевірки smoke_check.sh, який послідовно перевіряє конфігурацію, з'єднання з БД, версію схеми та коректність імпорту модулів системи.

```
(.venv) ~/Termux_SvetaNeBudit_bot $ ./scripts/smoke_check.sh
CONFIG_OK
STORAGE_OK
SCHEMA_OK:7
BOT_IMPORT_OK
Ручний runtime-smoke: запусти scripts/run_bot.sh і перевір Tel
egram-команди /start, /admin, /status, /schedule.
```

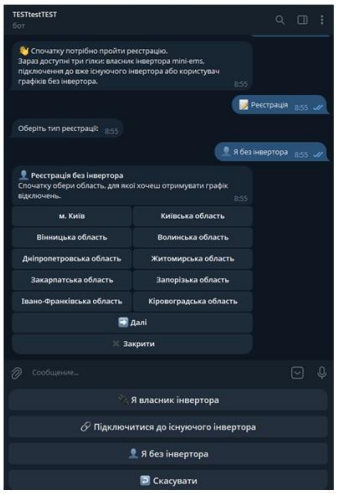
ⓘ Тестування системи проходило паралельно з розробкою весь час

Екранні форми: Telegram-взаємодія

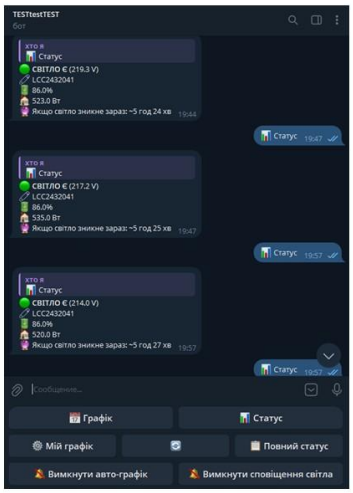
Усі ключові функції системи — меню, статус інвертора, персональний графік і сповіщення — доступні безпосередньо у Telegram без встановлення окремого застосунку.



Графік відключень
Погодинна таблиця з легендою стану світла



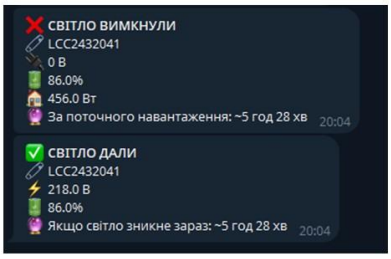
Реєстрація
Вибір типу ролі та налаштування регіону



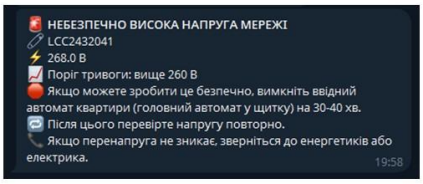
Статус інвертора
Напруга, заряд батареї та прогноз автономності

✓ Система успішно протестована в реальних умовах: миттєві сповіщення про зникнення світла, актуальний графік ДТЕК і статус інвертора — все в одному чаті.

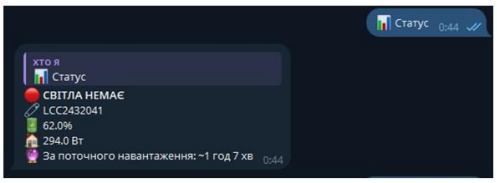
ЕКРАННІ ФОРМИ: ПОВІДОМЛЕННЯ СИСТЕМИ



Оперативні сповіщення
Повідомлення про зникнення та відновлення зовнішньої мережі формуються автоматично.

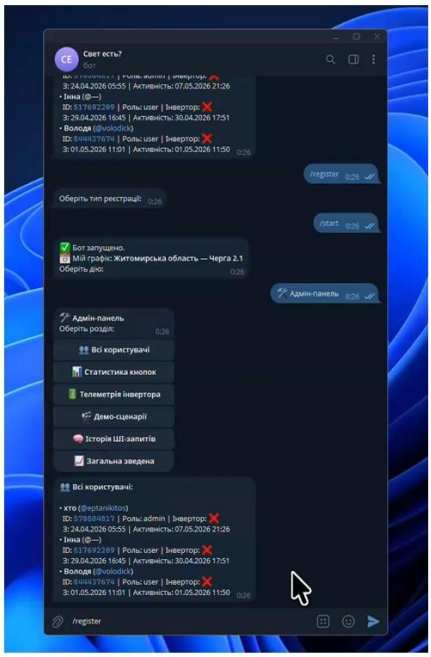


Тривоги за напругою
Система фіксує небезпечну високу напругу понад 260 В та низьку нижче 170 В.

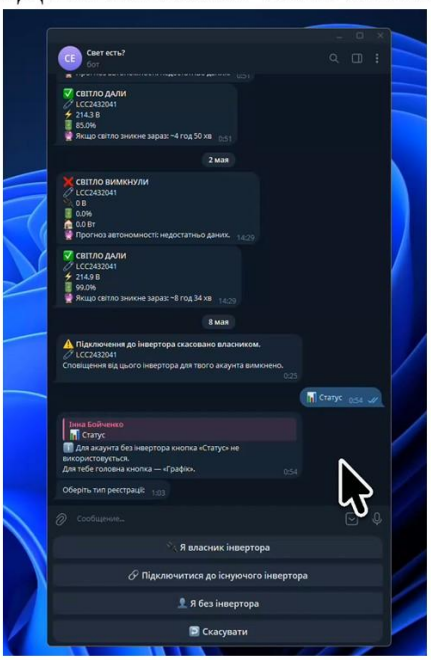


Прогнозування автономності
У статус додається орієнтовний час роботи від батареї за поточним або середнім навантаженням.

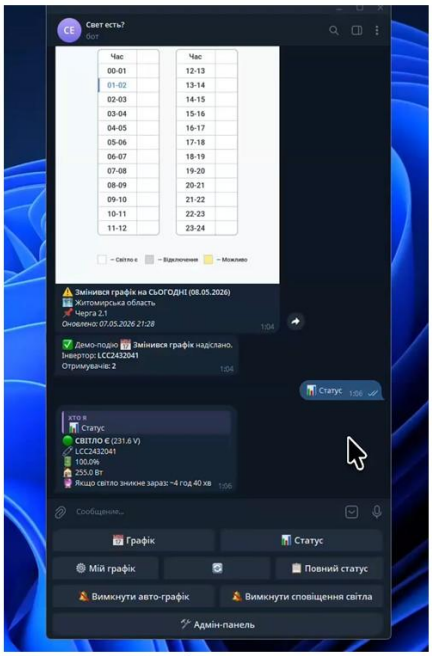
ЕКРАННІ ФОРМИ: ПОВІДОМЛЕННЯ СИСТЕМИ



Реєстрація власника



Підключення існуючого



Адмін панель

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ ТА ВИСНОВКИ

1. Бойченко Н.О., Аронов А.О. Особливості персоналізованого надання графіків відключень світла у телеграм-боті для моніторингу енергозабезпечення. // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. – К.: ДУІКТ, 2026, (подано до друку).

2. Бойченко Н.О., Аронов А.О. Спрощення взаємодії моніторингу інвертора через телеграм-бот з автоматичними сповіщеннями та зрозумілим поданням даних. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. – К.: ДУІКТ, 2026, (подано до друку).

ВИСНОВКИ

- → Досліджено предметну область побутового моніторингу електропостачання та резервного живлення;
- → Спроєктовано ролі користувачів, архітектуру системи, локальне сховище та основні сценарії роботи;
- → Реалізовано програмну систему на Python з aiogram, SQLite, SQLAlchemy та сервісним шаром для зовнішніх джерел;
- → Забезпечено автоматичні сповіщення, прогноз автономності, кешування графіків, шифроване зберігання облікових даних і розгортання в Termux.

Практичний результат

- → Створено придатну до побутового використання систему, яка централізує два різних джерела інформації про електропостачання у межах одного мобільного інтерфейсу.

Дякую за увагу!

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

```
import asyncio
import logging
import sys
from aiogram import Bot
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
from config import app_config
from handlers import admin_router, user_router
from services.ai_service import ai_service
from services.logging_dispatcher import
LoggingDispatcher
from services.monitoring import
monitor_registered_inverters_loop
from services.runtime_state import runtime_state
from services.schedule_runtime import
schedule_monitor_loop
from services.storage import storage_service
def load_users() -> None:
    power_ids, schedule_ids, admin_ids =
storage_service.load_runtime_sets()
    runtime_state.sync_users(power_ids, schedule_ids,
admin_ids)
async def main() -> None:
    storage_service.initialize()
    load_users()
    app_config.assert_required_settings()
    bot = Bot(
        token=app_config.telegram_token,

default=DefaultBotProperties(parse_mode=ParseMode.
HTML),
    )
    dispatcher = LoggingDispatcher()
    dispatcher.include_router(admin_router)
    dispatcher.include_router(user_router)
    print("🤖 Aiogram-бот запускається...")
    print(f"🧠 Модель ІІІІ: {ai_service.model_name}")
    print(f"📁 Дані: {app_config.data_dir}")
    print(f"🗄️ SQLite: {app_config.sqlite_db_path}")
    registered_monitor_task =
asyncio.create_task(monitor_registered_inverters_loop(b
ot))
    schedule_task =
asyncio.create_task(schedule_monitor_loop(bot))
    try:
        await dispatcher.start_polling(bot)
    finally:
        for task in (registered_monitor_task,
schedule_task):
            task.cancel()
        await asyncio.gather(registered_monitor_task,
schedule_task, return_exceptions=True)
        await bot.session.close()
if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO,
stream=sys.stdout)
    asyncio.run(main())
from __future__ import annotations

import asyncio
from datetime import datetime
from .autonomy import build_autonomy_forecast
from .inverter import inverter_service
from .runtime_state import runtime_state
from .storage import storage_service
USER_INVERTER_MONITOR_INTERVAL_SECON
DS = 5
GRID_VOLTAGE_LOW_ALERT_THRESHOLD =
170.0
GRID_VOLTAGE_HIGH_ALERT_THRESHOLD =
260.0
def _detect_grid_voltage_alert(snapshot) -> str | None:
    if not getattr(snapshot, "is_online", False):
        return None
    grid_volts = float(getattr(snapshot, "grid_volts", 0.0)
or 0.0)
    if grid_volts >
GRID_VOLTAGE_HIGH_ALERT_THRESHOLD:
        return "high"
    if grid_volts <
GRID_VOLTAGE_LOW_ALERT_THRESHOLD:
        return "low"
    return None
def _build_grid_status_message(snapshot, label: str |
None, forecast_line: str | None = None) -> str:
    device_label = label or "Ваш інвертор"
    if snapshot.is_online:
        message = (
            f"☑️ <b>СВІТЛО ДАЛИ</b>\n"
            f"🔗 {device_label}\n"
            f"⚡ {float(snapshot.grid_volts):.1f} В\n"
            f"🔋 {float(snapshot.battery_percentage):.1f}%"
        )
        if forecast_line:
            message += f"\n{forecast_line}"
        return message
    message = (
        f"❌ <b>СВІТЛО ВИМКНУЛИ</b>\n"
        f"🔗 {device_label}\n"
        f"🪫 0 В\n"
        f"🔋 {float(snapshot.battery_percentage):.1f}%\n"
        f"🏠 {float(snapshot.load_power):.1f} Вт"
    )
    if forecast_line:
        message += f"\n{forecast_line}"
    return message
async def
build_device_autonomy_forecast_line(device_id: int,
snapshot) -> str | None:
    telemetry_points = await asyncio.to_thread(
        storage_service.list_device_telemetry_points,
        device_id,
        8,
        6000,
    )
    forecast = build_autonomy_forecast(
```

```
        battery_percentage=getattr(snapshot,
"battery_percentage", None),
        load_power=getattr(snapshot, "load_power", None),
        is_online=bool(getattr(snapshot, "is_online",
False)),
        raw_row=getattr(snapshot, "raw_row", None),
        telemetry_points=telemetry_points,
    )
    return forecast.summary_line
def _build_voltage_alert_message(alert_kind: str,
snapshot, label: str | None) -> str:
    device_label = label or "Ваш інвертор"
    grid_volts = float(getattr(snapshot, "grid_volts", 0.0)
or 0.0)
    if alert_kind == "high":
        return (
            f"🚨 <b>НЕБЕЗПЕЧНО ВИСОКА НАПРУГА
МЕРЕЖІ</b>\n"
            f"🔗 {device_label}\n"
            f"⚡ {grid_volts:.1f} В\n"
            f"📈 Поріг тривоги: вище
{GRID_VOLTAGE_HIGH_ALERT_THRESHOLD:.0f
} В\n"
            f"🛑 Якщо можете зробити це безпечно,
вимкніть ввідний автомат квартири "
            f"(головний автомат у щитку) на 30-40 хв.\n"
            f"🔁 Після цього перевірте напругу
повторно.\n"
            f"📞 Якщо перенапруга не зникає, зверніться
до енергетиків або електрика."
        )
    return (
        f"🚨 <b>НЕБЕЗПЕЧНО НИЗЬКА НАПРУГА
МЕРЕЖІ</b>\n"
        f"🔗 {device_label}\n"
        f"⚡ {grid_volts:.1f} В\n"
        f"📉 Поріг тривоги: нижче
{GRID_VOLTAGE_LOW_ALERT_THRESHOLD:.0f}
В"
    )
async def monitor_registered_inverters_loop(bot) ->
None:
    print(f"🕯 Моніторинг зареєстрованих інверторів
активний
({USER_INVERTER_MONITOR_INTERVAL_SECO
NDS} с)!")
    while True:
        try:
            targets = await
asyncio.to_thread(storage_service.list_user_inverter_mo
nitor_targets)
            for target in targets:
                try:
                    data = await asyncio.to_thread(
                        inverter_service.get_data_for_account,
                        target["site_username"],
                        target["account_id"],
                        target["power_id"],
                        target["inverter_id"],
                    )
                    snapshot =
inverter_service.build_snapshot(data)
                    if not snapshot:
                        continue

runtime_state.record_load(snapshot.load_power)
                    runtime_state.set_owner_snapshot(
                        target["device_id"],
                        snapshot,
                        target.get("label"),
                    )
                    await asyncio.to_thread(

storage_service.record_device_telemetry_sample,
                        target["device_id"],
                        snapshot,
                        None,
                        target.get("label"),
                    )
                    grid_changed =
runtime_state.update_owner_grid_status(target["device_i
d"], snapshot.is_online)
                    voltage_alert_kind =
_detect_grid_voltage_alert(snapshot)
                    new_voltage_alert_kind =
runtime_state.update_owner_voltage_alert_state(
                        target["device_id"],
                        voltage_alert_kind,
                    )
                    if grid_changed:
                        current_watchers = await
asyncio.to_thread(

storage_service.list_active_device_watcher_chat_ids,
                            target["device_id"],
                        )
                        forecast_line = await
build_device_autonomy_forecast_line(target["device_id"
], snapshot)
                        message =
_build_grid_status_message(snapshot,
target.get("label"), forecast_line)
                        for chat_id in current_watchers:
                            try:
                                await bot.send_message(chat_id,
message)
                            except Exception as exc:
                                print(f"⚠️ Не вдалося надіслати
повідомлення юзеру {chat_id}: {exc}")
                    if new_voltage_alert_kind:
                        current_watchers = await
asyncio.to_thread(

storage_service.list_active_device_watcher_chat_ids,
                            target["device_id"],
                        )
                        voltage_message =
_build_voltage_alert_message(
                            new_voltage_alert_kind,
                            snapshot,
                            target.get("label"),
                        )
```

```
                    for chat_id in current_watchers:
                        try:
                            await bot.send_message(chat_id,
voltage_message)
                        except Exception as exc:
                            print(f" ⚠️ Не вдалося надіслати
тривогу по напрузі юзеру {chat_id}: {exc}")
                print(

f"[{datetime.now().strftime('%H:%M:%S')}] "
                    f"[user-monitor]
device:{target['device_id']}
watchers:{len(target.get('watcher_chat_ids', []))} "
                    f"V:{snapshot.grid_volts} "
                    f"Bat:{snapshot.battery_percentage}% "
                    f"Load:{snapshot.load_power}W"
                )
            except Exception as exc:
                print(f" ⚠️ Помилка моніторингу
пристрою {target.get('device_id')}: {exc}")
        except Exception as exc:

print(f"[{datetime.now().strftime('%H:%M:%S')}] ❌
Помилка цикла user-monitoring: {exc}")
        await
asyncio.sleep(USER_INVERTER_MONITOR_INTER
VAL_SECONDS)
```

```
from __future__ import annotations
import asyncio
from dataclasses import dataclass
from datetime import datetime, timedelta
import requests
from aiogram.types import BufferedInputFile
from parsers import
fetch_region_archive_schedule_map_safe,
fetch_region_schedule_map_safe
from renderers import generate_schedule_image
from .runtime_state import BufferedScheduleSnapshot,
runtime_state
from .storage import storage_service
SCHEDULE_MONITOR_INTERVAL_SECONDS =
600
SCHEDULE_ADMIN_ALERT_COOLDOWN_SECON
DS = 6 * 3600
SCHEDULE_SITE_UNAVAILABLE_TEXT = " ⚠️
Сайт графіків тимчасово недоступний."
SCHEDULE_SITE_UNAVAILABLE_WITH_CACHE_
TEXT = " ⚠️ Сайт графіків тимчасово недоступний.
Показую останній збережений графік."
SCHEDULE_STRUCTURE_CHANGED_TEXT = " ⚠️
Структура сторінки змінилася. Потрібне оновлення
парсера."
SCHEDULE_REGION_HAS_NO_GROUPS_TEXT =
" ⚠️ Для цієї області зараз немає доступного
графіка."
SCHEDULE_QUEUE_NOT_FOUND_TEXT = " ⚠️
Графік для цієї групи поки не знайдено."
SCHEDULE_TOMORROW_MISSING_TEXT = "Не
знайшов графік на завтра 😔"
SCHEDULE_TOMORROW_MISSING_NOTICE_CO
OLDOWN_SECONDS = 12 * 3600
@dataclass(frozen=True)
class ScheduleFetchOutcome:
    snapshot: BufferedScheduleSnapshot | None
    warning_text: str | None = None
    archive_date_label: str | None = None
def _create_schedule_session() -> requests.Session:
    session = requests.Session()
    session.headers.update(
        {
            "User-Agent": "Mozilla/5.0 (Windows NT 10.0;
Win64; x64)",
            "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9"
,
            "Accept-Language": "uk-UA,uk;q=0.9",
        }
    )
    return session
async def _render_photo(half_hours, highlight: bool,
filename: str) -> BufferedInputFile:
    image_stream = await
asyncio.to_thread(generate_schedule_image, half_hours,
highlight)
    return BufferedInputFile(image_stream.getvalue(),
filename=filename)
def _schedule_date_label(day_key: str) -> tuple[str, str,
bool]:
    now = datetime.now()
    if day_key == "tomorrow":
        return "ЗАВТРА", (now +
timedelta(days=1)).strftime("%d.%m.%Y"), False
    return "СЬОГОДНІ", now.strftime("%d.%m.%Y"),
True
async def _store_schedule_snapshot(
    day_key: str,
    region_slug: str,
    region_name: str | None,
    queue_code: str,
    queue_label: str | None,
    source_url: str,
    half_hours,
    update_label: str | None,
) -> None:
    if not half_hours:
        return
    try:
        await asyncio.to_thread(
            storage_service.record_schedule_snapshot,
            day_key,
            source_url,
            half_hours,
            update_label,
            queue_code,
            region_slug,
            region_name,
            queue_label,
        )
    except Exception as exc:
```

```
        print(f" ⚠️ Не вдалося зберегти snapshot графіка ({region_slug}/{queue_code}/{day_key}): {exc}")
def _schedule_changed(previous: BufferedScheduleSnapshot | None, half_hours, update_label: str | None) -> bool:
    if previous is None:
        return False
    return previous.half_hours != list(half_hours or []) or previous.update_label != update_label
def _warning_for_schedule_error(error_kind: str | None, *, has_cached_snapshot: bool, is_admin: bool) -> str | None:
    if error_kind == "site_unavailable":
        return SCHEDULE_SITE_UNAVAILABLE_WITH_CACHE_TEXT if has_cached_snapshot else SCHEDULE_SITE_UNAVAILABLE_TEXT
    if error_kind == "structure_changed":
        if is_admin:
            return SCHEDULE_STRUCTURE_CHANGED_TEXT
        return SCHEDULE_SITE_UNAVAILABLE_WITH_CACHE_TEXT if has_cached_snapshot else SCHEDULE_SITE_UNAVAILABLE_TEXT
    if error_kind == "region_has_no_groups":
        return SCHEDULE_REGION_HAS_NO_GROUPS_TEXT
    if error_kind == "queue_not_found":
        return SCHEDULE_QUEUE_NOT_FOUND_TEXT
    return " ⚠️ Не вдалося оновити графік. Спробуй трохи пізніше."
def _day_specific_warning_text(warning_text: str | None, day_key: str) -> str | None:
    if not warning_text:
        return None
    if day_key == "tomorrow" and warning_text in {SCHEDULE_QUEUE_NOT_FOUND_TEXT, SCHEDULE_REGION_HAS_NO_GROUPS_TEXT}:
        return SCHEDULE_TOMORROW_MISSING_TEXT
    return warning_text
async def _restore_schedule_snapshot_from_storage(schedule_profile: dict, day_key: str) -> BufferedScheduleSnapshot | None:
    snapshot_payload = await asyncio.to_thread(
        storage_service.get_latest_schedule_snapshot,
        day_key,
        schedule_profile["region_slug"],
        schedule_profile["queue_code"],
    )
    if snapshot_payload is None:
        return None
    runtime_state.set_schedule_snapshot(
        schedule_profile["region_slug"],
        schedule_profile["queue_code"],
        day_key,
        snapshot_payload["half_hours"],
        snapshot_payload.get("update_label"),
        snapshot_payload.get("source_url") or "",
        region_name=snapshot_payload.get("region_name"),
        queue_label=snapshot_payload.get("queue_label"),
    )
    return runtime_state.get_schedule_snapshot(
        schedule_profile["region_slug"],
        schedule_profile["queue_code"],
        day_key,
    )
async def _resolve_previous_day_archive_snapshot(schedule_profile: dict) -> ScheduleFetchOutcome | None:
    archive_date = (datetime.now() - timedelta(days=1)).date()
    fetch_result = await asyncio.to_thread(
        fetch_region_archive_schedule_map_safe,
        schedule_profile["region_slug"],
        archive_date,
    )
    if fetch_result.error is not None:
        return None
    entry = fetch_result.entries.get(schedule_profile["queue_code"])
    if entry is None:
        return None
    archive_date_label = archive_date.strftime("%d.%m.%Y")
    snapshot = BufferedScheduleSnapshot(
        region_slug=schedule_profile["region_slug"],
        region_name=fetch_result.region_name or schedule_profile["region_name"],
        queue_code=schedule_profile["queue_code"],
        queue_label=entry.queue_label or schedule_profile["queue_label"],
        day_key="today",
        half_hours=list(entry.half_hours),
        update_label=entry.update_label,
        source_url=entry.source_url or fetch_result.source_url,
        captured_at=datetime.now(),
    )
    return ScheduleFetchOutcome(
        snapshot=snapshot,
        warning_text=f"ℹ️ Графік на сьогодні ще не опублікований. Показую останній доступний графік за {archive_date_label}.",
        archive_date_label=archive_date_label,
    )
async def _notify_admins_about_schedule_structure(bot, region_slug: str, day_key: str, error) -> None:
    if error is None or error.kind != "structure_changed" or not runtime_state.admin_ids:
        return
    alert_key = f"{region_slug}:{day_key}:{error.kind}"
    if not runtime_state.should_send_schedule_admin_alert(alert_key, SCHEDULE_ADMIN_ALERT_COOLDOWN_SECONDS):
        return
    text = (
```

```
        "⚠️ <b>Bezsvitla: структура сторінки змінилася</b>\n"
        f"🗺️ Регіон: <b>{region_slug}</b>\n"
        f"🗓️ День: <b>{'сьогодні' if day_key == 'today' else 'завтра'}</b>\n"
        f"🔗 URL: <code>{error.url}</code>"
    )
    if error.details:
        text += f"\n🧩 Деталі: <code>{error.details[:250]}</code>"
    for admin_id in sorted(runtime_state.admin_ids):
        try:
            await bot.send_message(admin_id, text)
        except Exception as exc:
            print(f"⚠️ Не вдалося надіслати admin-alert про парсер графіка (chat={admin_id}): {exc}")
async def _notify_watchers_tomorrow_missing(
    bot,
    region_slug: str,
    region_name: str,
    queue_target: dict,
) -> None:
    watcher_chat_ids = queue_target.get("watcher_chat_ids") or []
    if not watcher_chat_ids:
        return
    tomorrow_key = (datetime.now() + timedelta(days=1)).strftime("%Y-%m-%d")
    notice_key = f"tomorrow-missing:{region_slug}:{queue_target['queue_code']}:{tomorrow_key}"
    if not runtime_state.should_send_schedule_notice(notice_key, SCHEDULE_TOMORROW_MISSING_NOTICE_COOLDOWN_SECONDS):
        return
    text = (
        f"{SCHEDULE_TOMORROW_MISSING_TEXT}\n"
        f"🗺️ {region_name}\n"
        f"📌 {queue_target.get('queue_label') or queue_target['queue_code']}"
    )
    for user_id in watcher_chat_ids:
        try:
            await bot.send_message(user_id, text)
        except Exception as exc:
            print(f"⚠️ Не вдалось надіслати повідомлення про відсутній графік на завтра юзеру {user_id}: {exc}")
async def safe_update_status(bot, chat_id: int, msg_id: int, text: str) -> None:
    try:
        await bot.edit_message_text(text=text, chat_id=chat_id, message_id=msg_id)
    except Exception as exc:
        print(f"⚠️ Не вдалось оновити статус (chat={chat_id}, msg={msg_id}): {exc}")
async def broadcast_schedule_change(
    bot,
    watcher_chat_ids: list[int],
    region_name: str,
    queue_label: str,
    day_key: str,
    half_hours,
    update_time: str | None,
    action_type: str = "Оновлено",
) -> None:
    if not watcher_chat_ids:
        return
    day_type, date_str, highlight = _schedule_date_label(day_key)
    photo = await _render_photo(half_hours, highlight, f"schedule-{day_key}.png")
    if action_type == "Додано":
        caption = (
            f"🎉 <b>З'явився графік на {day_type} ({date_str})</b>\n"
            f"🗺️ {region_name}\n"
            f"📌 {queue_label}\n"
            f"<i>Оновлено: {update_time or 'Невідомо'}</i>"
        )
    else:
        caption = (
            f"⚠️ <b>Змінився графік на {day_type} ({date_str})</b>\n"
            f"🗺️ {region_name}\n"
            f"📌 {queue_label}\n"
            f"<i>Оновлено: {update_time or 'Невідомо'}</i>"
        )
    for user_id in watcher_chat_ids:
        try:
            await bot.send_photo(user_id, photo=photo, caption=caption)
        except Exception as exc:
            print(f"⚠️ Не вдалось надіслати графік юзеру {user_id}: {exc}")
async def _poll_region_day(
    bot,
    region_target: dict,
    day_key: str,
    session: requests.Session,
    *,
    notify_tomorrow_missing: bool = True,
) -> None:
    region_slug = region_target["region_slug"]
    region_name = region_target.get("region_name") or region_slug
    fetch_result = await asyncio.to_thread(fetch_region_schedule_map_safe, region_slug, day_key, session)
    if fetch_result.error is not None:
        print(
            f"⚠️ Графік {region_slug}/{day_key}: {fetch_result.error.kind} "
            f"({fetch_result.error.url})"
        )
        if day_key == "tomorrow" and fetch_result.error.kind == "region_has_no_groups" and notify_tomorrow_missing:
```

```
            for queue_target in
region_target["queues"].values():
                await _notify_watchers_tomorrow_missing(
                    bot,
                    region_slug,
                    region_name,
                    queue_target,
                )
        await
_notify_admins_about_schedule_structure(bot,
region_slug, day_key, fetch_result.error)
        return
    resolved_region_name = fetch_result.region_name or
region_name
    for queue_code, queue_target in
region_target["queues"].items():
        entry = fetch_result.entries.get(queue_code)
        if entry is None:
            print(
                f"⚠️ Графік
{region_slug}/{queue_code}/{day_key}: "
                "чергу не знайдено на поточній сторінці,
зберігаю попередній snapshot"
            )
            if day_key == "tomorrow" and
notify_tomorrow_missing:
                await _notify_watchers_tomorrow_missing(
                    bot,
                    region_slug,
                    resolved_region_name,
                    queue_target,
                )
            continue
        previous =
runtime_state.get_schedule_snapshot(region_slug,
queue_code, day_key)
        resolved_queue_label = entry.queue_label or
queue_target.get("queue_label") or f"Черга
{queue_code}"
        await _store_schedule_snapshot(
            day_key,
            region_slug,
            resolved_region_name,
            queue_code,
            resolved_queue_label,
            entry.source_url or fetch_result.source_url,
            entry.half_hours,
            entry.update_label,
        )
        runtime_state.set_schedule_snapshot(
            region_slug,
            queue_code,
            day_key,
            entry.half_hours,
            entry.update_label,
            entry.source_url or fetch_result.source_url,
            region_name=resolved_region_name,
            queue_label=resolved_queue_label,
        )
        if _schedule_changed(previous, entry.half_hours,
entry.update_label):
            await broadcast_schedule_change(
                bot=bot,

watcher_chat_ids=queue_target["watcher_chat_ids"],
                region_name=resolved_region_name,
                queue_label=resolved_queue_label,
                day_key=day_key,
                half_hours=entry.half_hours,
                update_time=entry.update_label,
                action_type="Оновлено",
            )
async def schedule_monitor_loop(bot) -> None:
    print("🕒 Aiogram-моніторинг графіків запущено
(раз на 10 хв, по регіонах)")
    warmup_cycle = True
    while True:
        try:
            monitor_targets = await
asyncio.to_thread(storage_service.list_schedule_monitor
_targets)
            current_time =
datetime.now().strftime("%H:%M:%S")
            if not monitor_targets:
                print(f"[{current_time}] 🔄 Графіки:
активних регіональних підписок немає")
            else:
                total_queues = sum(len(item["queues"]) for
item in monitor_targets)
                print(f"[{current_time}] 🔄 Перевірка
графіків: регіонів={len(monitor_targets)}
груп={total_queues}")
                for region_target in monitor_targets:
                    session = _create_schedule_session()
                    try:
                        await _poll_region_day(
                            bot,
                            region_target,
                            "today",
                            session,
                            notify_tomorrow_missing=False,
                        )
                        await _poll_region_day(
                            bot,
                            region_target,
                            "tomorrow",
                            session,
                            notify_tomorrow_missing=False,
                        )
                    finally:
                        session.close()
        except Exception as exc:

print(f"[{datetime.now().strftime('%H:%M:%S')}] ❌
Помилка моніторингу графіків: {exc}")
        warmup_cycle = False
        await
asyncio.sleep(SCHEDULE_MONITOR_INTERVAL_S
ECONDS)
async def
_fetch_live_profile_snapshots(schedule_profile: dict) ->
tuple[BufferedScheduleSnapshot | None,
BufferedScheduleSnapshot | None]:
```

```
    today_outcome = await
_resolve_schedule_snapshot(schedule_profile, "today")
    tomorrow_outcome = await
_resolve_schedule_snapshot(schedule_profile,
"tomorrow")
    return today_outcome.snapshot,
tomorrow_outcome.snapshot
async def _resolve_schedule_snapshot(
    schedule_profile: dict,
    day_key: str,
    *,
    allow_runtime_buffer: bool = True,
    is_admin: bool = False,
) -> ScheduleFetchOutcome:
    region_slug = schedule_profile["region_slug"]
    region_name = schedule_profile["region_name"]
    queue_code = schedule_profile["queue_code"]
    queue_label = schedule_profile["queue_label"]
    if allow_runtime_buffer:
        runtime_snapshot =
runtime_state.get_schedule_snapshot(region_slug,
queue_code, day_key)
        if runtime_snapshot is not None:
            return ScheduleFetchOutcome(runtime_snapshot,
None)
    session = _create_schedule_session()
    fetch_result = await
asyncio.to_thread(fetch_region_schedule_map_safe,
region_slug, day_key, session)
    if fetch_result.error is not None:
        if day_key == "today" and fetch_result.error.kind
== "region_has_no_groups":
            archive_outcome = await
_resolve_previous_day_archive_snapshot(schedule_profi
le)
            if archive_outcome is not None:
                return archive_outcome
        snapshot = await
_restore_schedule_snapshot_from_storage(schedule_pro
file, day_key)
        return ScheduleFetchOutcome(
            snapshot,
            _warning_for_schedule_error(
                fetch_result.error.kind,
                has_cached_snapshot=snapshot is not None,
                is_admin=is_admin,
            ),
        )
    entry = fetch_result.entries.get(queue_code)
    if entry is None:
        if day_key == "today":
            archive_outcome = await
_resolve_previous_day_archive_snapshot(schedule_profi
le)
            if archive_outcome is not None:
                return archive_outcome
        snapshot = await
_restore_schedule_snapshot_from_storage(schedule_pro
file, day_key)
        return ScheduleFetchOutcome(
            snapshot,
            _warning_for_schedule_error(
                "queue_not_found",
                has_cached_snapshot=snapshot is not None,
                is_admin=is_admin,
            ),
        )
    resolved_region_name = fetch_result.region_name or
region_name
    resolved_queue_label = entry.queue_label or
queue_label
    await _store_schedule_snapshot(
        day_key,
        region_slug,
        resolved_region_name,
        queue_code,
        resolved_queue_label,
        entry.source_url or fetch_result.source_url,
        entry.half_hours,
        entry.update_label,
    )
    runtime_state.set_schedule_snapshot(
        region_slug,
        queue_code,
        day_key,
        entry.half_hours,
        entry.update_label,
        entry.source_url or fetch_result.source_url,
        region_name=resolved_region_name,
        queue_label=resolved_queue_label,
    )
    return ScheduleFetchOutcome(
        runtime_state.get_schedule_snapshot(region_slug,
queue_code, day_key),
        None,
    )
async def
resolve_demo_schedule_snapshot(schedule_profile: dict)
-> tuple[str | None, BufferedScheduleSnapshot | None]:
    for day_key in ("tomorrow", "today"):
        outcome = await
_resolve_schedule_snapshot(schedule_profile, day_key,
is_admin=True)
        if outcome.snapshot is not None:
            return day_key, outcome.snapshot
    return None, None
async def _send_schedule_photo(
    bot,
    chat_id: int,
    schedule_profile: dict,
    snapshot: BufferedScheduleSnapshot,
    day_key: str,
    archive_date_label: str | None = None,
) -> None:
    day_type, date_str, highlight =
_schedule_date_label(day_key)
    if archive_date_label:
        highlight = False
    photo = await _render_photo(snapshot.half_hours,
highlight, f"schedule-{day_key}.png")
    if archive_date_label:
        caption = (
            f"🗓 <b>Останній доступний графік
({archive_date_label})</b>\n"
```

```
            f"🗺️ {snapshot.region_name or
schedule_profile['region_name']}\n"
            f"📌 {snapshot.queue_label or
schedule_profile['queue_label']}\n"
            f"<i>Оновлено: {snapshot.update_label or
'Невідомо'}</i>"
        )
    else:
        caption = (
            f"🗓️ <b>Графік на {day_type}
({date_str})</b>\n"
            f"🗺️ {snapshot.region_name or
schedule_profile['region_name']}\n"
            f"📌 {snapshot.queue_label or
schedule_profile['queue_label']}\n"
            f"<i>Оновлено: {snapshot.update_label or
'Невідомо'}</i>"
        )
    await bot.send_photo(chat_id, photo=photo,
caption=caption)


async def fetch_and_send_schedule(bot, chat_id: int,
status_msg_id: int) -> None:
    await safe_update_status(bot, chat_id, status_msg_id,
"⏳ <b>Завантажую графіки на сьогодні та
завтра...</b>")

    schedule_profile = await
asyncio.to_thread(storage_service.get_user_schedule_pr
ofile, chat_id)
    if schedule_profile is None:
        await safe_update_status(
            bot,
            chat_id,
            status_msg_id,
            "⚠️ Спочатку обери область і групу для свого
графіка.",
        )
        return
    try:
        is_admin = chat_id in runtime_state.admin_ids
        today_outcome = await
_resolve_schedule_snapshot(schedule_profile, "today",
is_admin=is_admin)
        tomorrow_outcome = await
_resolve_schedule_snapshot(schedule_profile,
"tomorrow", is_admin=is_admin)
        today_snapshot = today_outcome.snapshot
        tomorrow_snapshot = tomorrow_outcome.snapshot
        if today_snapshot is None:
            failure_text = today_outcome.warning_text or
SCHEDULE_QUEUE_NOT_FOUND_TEXT
            await safe_update_status(bot, chat_id,
status_msg_id, failure_text)
            return
        await bot.delete_message(chat_id, status_msg_id)
        today_warning =
_day_specific_warning_text(today_outcome.warning_te
xt, "today")
        if today_warning:
            await bot.send_message(chat_id, today_warning)
        await _send_schedule_photo(
            bot,
            chat_id,
            schedule_profile,
            today_snapshot,
            "today",

archive_date_label=today_outcome.archive_date_label,
        )
        if tomorrow_snapshot is not None:
            tomorrow_warning =
_day_specific_warning_text(tomorrow_outcome.warnin
g_text, "tomorrow")
            if tomorrow_warning:
                await bot.send_message(chat_id,
tomorrow_warning)
            await _send_schedule_photo(
                bot,
                chat_id,
                schedule_profile,
                tomorrow_snapshot,
                "tomorrow",

archive_date_label=tomorrow_outcome.archive_date_la
bel,
            )
        else:
            await bot.send_message(
                chat_id,

_day_specific_warning_text(tomorrow_outcome.warnin
g_text, "tomorrow") or
SCHEDULE_TOMORROW_MISSING_TEXT,
            )
    except Exception as exc:
        print(f"❌ Помилка fetch_and_send_schedule
(chat={chat_id}): {exc}")
        await safe_update_status(bot, chat_id,
status_msg_id, f"❌ <b>Технічна
помилка:</b>\n<code>{exc}</code>")
```