

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ПОГОДЖЕНО
Декан факультету

_____Ігор БОЛБОТ
(підпис)

«___» _____2026 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних
наук

_____Белла ГОЛУБ
(підпис)

«___» _____2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Програмне забезпечення інформаційної системи обліку
телеметрії автомобіля»**

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми

К.Т.Н., доцент

_____ **Ганна ВАЙГАНГ**
(підпис)

**Керівник бакалаврської
кваліфікаційної роботи**

ст.викладач

_____ **Георгій Бородкін**
(підпис)

Виконав

_____ **Валентин Возний**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

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

«___» _____ 2025 р.

ЗАВДАННЯ ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ЗДОБУВАЧУ

Возному Валентину Володимировичу

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

**« Програмне забезпечення інформаційної системи обліку телеметрії
автомобіля»**

затверджена наказом від «19 грудня» 2025 р. № 3204«С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи:

Технічні характеристики та регламент обслуговування відповідних типів авто.

Перелік питань, що підлягають дослідженню:

- Аналіз предметної області та порівняльне дослідження існуючих систем агроменеджменту і симуляції сільськогосподарських процесів.
- Проектування архітектури бази даних та графічного інтерфейсу управління полями й агроресурсами.
- Програмна реалізація математичних моделей росту культур, бізнес-логіки агротехнічних операцій та калькуляції фінансових показників.
- Тестування функціональних можливостей системи та організація взаємодії програмних компонентів із реляційною базою даних.

Перелік графічних документів (за необхідності).

Дата видачі завдання «25» грудня 2025 р.

**Керівник бакалаврської
кваліфікаційної роботи**

_____ **Георгій Бородкін**
(підпис)

Завдання взяв до виконання

_____ **Валентин Возний**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ	5
ВСТУП.....	7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1. Опис предметної області.....	10
1.2. Аналіз існуючих рішень у сфері обліку телеметрії авто	13
1.3. Моделювання предметної області	18
1.4. Аналіз вимог розроблюваної системи	22
1.5. Постановка завдання	25
2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	28
2.1 Логічна модель даних у вигляді ER-діаграми	28
2.2 Діаграма класів і кооперації	31
2.3 Діаграма компонентів	37
2.4 Діаграма пакетів	40
2.5 Висновки до другого розділу	44
3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ ТА РОЗРОБЛЕННЯ АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ	47
3.1 Обґрунтування вибору технологічних засобів та інструментів реалізації.....	47
3.2 Фізична модель даних	49
3.3 Алгоритмізація програмних модулів системи.....	53
3.4 Висновки до третього розділу	60
4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ	62

4.1 План тестування програмних модулів та методика оцінювання результатів.....	62
4.2 Тестування апаратно-програмної системи моніторингу стану транспортного засобу	64
4.3 Результати тестування та аналіз ефективності системи	68
4.4 Висновки до четвертого розділу	70
ВИСНОВКИ.....	73
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	76
Додатки.....	79

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

1. API – прикладний програмний інтерфейс (Application Programming Interface)
2. CAN – контролерна мережа транспортного засобу (Controller Area Network)
3. CSV – формат табличних даних із розділенням значень комами (Comma-Separated Values)
4. DTC – діагностичний код несправності транспортного засобу (Diagnostic Trouble Code)
5. DTO – об’єкт передавання даних між модулями програми (Data Transfer Object)
6. ER-діаграма – діаграма «сутність–зв’язок» (Entity–Relationship Diagram)
7. GPS – глобальна система позиціонування (Global Positioning System)
8. GUI – графічний інтерфейс користувача (Graphical User Interface)
9. IMEI – міжнародний ідентифікатор мобільного обладнання (International Mobile Equipment Identity)
10. JSON – текстовий формат обміну структурованими даними (JavaScript Object Notation)
11. KPI – ключовий показник ефективності (Key Performance Indicator)
12. OBD-II – бортова система діагностики автомобіля другого покоління (On-Board Diagnostics II)
13. PDF – формат електронного документа (Portable Document Format)
14. PID – ідентифікатор параметра OBD-II (Parameter ID)
15. PyQt6 – бібліотека Python для створення графічних інтерфейсів на основі Qt 6
16. Qt – кросплатформний фреймворк для розроблення графічних інтерфейсів
17. RBAC – керування доступом на основі ролей (Role-Based Access Control)
18. RPM – кількість обертів двигуна за хвилину (Revolutions Per Minute)
19. SQLite – вбудована файлова система керування базами даних
20. SQL – мова структурованих запитів до баз даних (Structured Query Language)
21. UML – уніфікована мова моделювання (Unified Modeling Language)

- 22.UUID – універсальний унікальний ідентифікатор (Universally Unique Identifier)
- 23.VIN – ідентифікаційний номер транспортного засобу (Vehicle Identification Number)
- 24.БД – база даних
- 25.ІС – інформаційна система
- 26.ПЗ – програмне забезпечення
- 27.СКБД – система керування базами даних
- 28.ТО – технічне обслуговування

ВСТУП

Інтенсивний розвиток транспортної інфраструктури, зростання кількості підключених транспортних засобів та підвищення вимог до безпеки і ефективності їх експлуатації актуалізують потребу у впровадженні сучасних інформаційних технологій для збору та аналізу телеметричних даних. Сучасні автомобілі оснащуються великою кількістю датчиків, що генерують значні обсяги інформації про технічний стан агрегатів, параметри руху, витрати пального, екологічні показники та поведінку водія. Традиційні підходи до обліку таких даних здебільшого базуються на періодичній діагностиці або локальному зчитуванні інформації, що не забезпечує безперервного моніторингу, своєчасного виявлення відхилень та комплексного аналізу експлуатаційних характеристик. У цих умовах розроблення програмного забезпечення інформаційної системи обліку телеметрії авто набуває особливої актуальності, оскільки дозволяє автоматизувати процес збору, передавання, збереження та аналітичного оброблення телеметричних потоків у режимі реального часу.

Створення програмного забезпечення інформаційної системи обліку телеметрії авто є важливою науково-прикладною задачею, спрямованою на підвищення ефективності управління транспортними засобами, оптимізацію витрат на технічне обслуговування, зниження ризиків аварійних ситуацій та формування аналітичної бази для прийняття управлінських рішень. Реалізація такої системи забезпечує централізований облік телеметричних даних, інтеграцію з бортовими пристроями (OBD-II, CAN-шина), формування звітів, побудову прогнозних моделей зношування вузлів та підтримку механізмів попереджувальної діагностики.

Метою дослідження є проектування та розробка програмного забезпечення інформаційної системи обліку телеметрії авто, здатної забезпечувати автоматизований збір, збереження, оброблення та аналітичний аналіз телеметричних даних транспортних засобів у режимі реального часу.

Для досягнення поставленої мети сформульовано такі основні **завдання**:

- здійснити системний аналіз предметної області та визначити функціональні і нефункціональні вимоги до інформаційної системи;
- розробити архітектуру програмного забезпечення з урахуванням потокового характеру телеметричних даних;
- спроектувати модель даних та структуру сховища для збереження великих обсягів телеметричної інформації;
- реалізувати модулі збору та передавання даних із бортових пристроїв транспортного засобу;
- розробити механізми оброблення, агрегації та візуалізації телеметричних показників;
- забезпечити інтеграцію системи з зовнішніми сервісами моніторингу, GPS-навігації та аналітичними модулями;
- реалізувати підсистему формування звітів і попереджень про критичні відхилення параметрів;
- провести тестування продуктивності, масштабованості та надійності функціонування системи в умовах реальних експлуатаційних навантажень;
- оцінити ефективність впровадження системи для підвищення якості технічного обслуговування та управління автопарком.

Об'єктом дослідження є процес моніторингу та обліку телеметричних параметрів транспортних засобів у інформаційних системах керування автотранспортом.

Предметом дослідження є методи, алгоритми та програмні засоби збору, збереження, передавання й аналітичного оброблення телеметричних даних автомобіля в умовах потокового надходження інформації.

Методологічною основою дослідження є методи системного аналізу, архітектурного проєктування інформаційних систем, моделювання потоків даних, технології оброблення великих масивів даних, принципи побудови мікросервісної та клієнт-серверної архітектур, а також інструменти структурного та об'єктно-орієнтованого моделювання (UML, DFD, ER-моделювання).

Наукова новизна роботи полягає у розробленні програмного забезпечення інтегрованої інформаційної системи обліку телеметрії авто, що забезпечує узгоджену взаємодію бортових джерел даних, серверної інфраструктури та аналітичних модулів; у формуванні адаптивної архітектури оброблення поточкових телеметричних даних із забезпеченням масштабованості та відмовостійкості; у вдосконаленні механізмів агрегації та інтерпретації показників технічного стану транспортного засобу; а також у розробленні методики комплексної оцінки ефективності функціонування системи в умовах реального використання автотранспорту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області

Сучасні транспортні засоби функціонують як складні кіберфізичні системи, що генерують значні обсяги телеметричних даних у процесі експлуатації. Параметри роботи двигуна, трансмісії, паливної системи, електронних блоків керування, а також навігаційні та поведінкові характеристики водія формуються у вигляді безперервних потоків даних, що передаються через внутрішні шини обміну (CAN, LIN) та діагностичні інтерфейси (OBD-II). У таких умовах критично важливим стає створення інформаційної системи, здатної забезпечити централізований облік, збереження та аналітичне оброблення телеметрії в режимі реального часу. На рис. 1.1 подано узагальнену архітектуру потоків даних та компонентів інформаційної системи обліку телеметрії авто.

Отримання вхідної інформації здійснюється з бортових джерел даних, до складу яких входять OBD-II адаптери, CAN-адаптери, GPS/IMU-модулі та інші сенсорні підсистеми транспортного засобу. Вони формують сирі телеметричні пакети, що містять ідентифікатори пристроїв, часові мітки, коди PID/CAN-ID, значення параметрів та службову інформацію контролю цілісності. Ці пакети передаються до серверної частини системи через мобільні або бездротові канали зв'язку, де здійснюється їх приймання, валідація, нормалізація та збагачення контекстними даними (наприклад, географічними координатами).

У підсистемі збору та нормалізації відбувається перетворення сирих кадрів протоколів у структуровані телеметричні точки, придатні для збереження у базі даних часових рядів. На цьому етапі виконується фільтрація дублікатів, узгодження одиниць вимірювання, синхронізація часових міток, а також первинна перевірка коректності значень. Нормалізовані дані зберігаються в

оперативному сховищі телеметрії, оптимізованому для роботи з ВИСОКОЧАСТОТНИМИ ПОТОКАМИ.

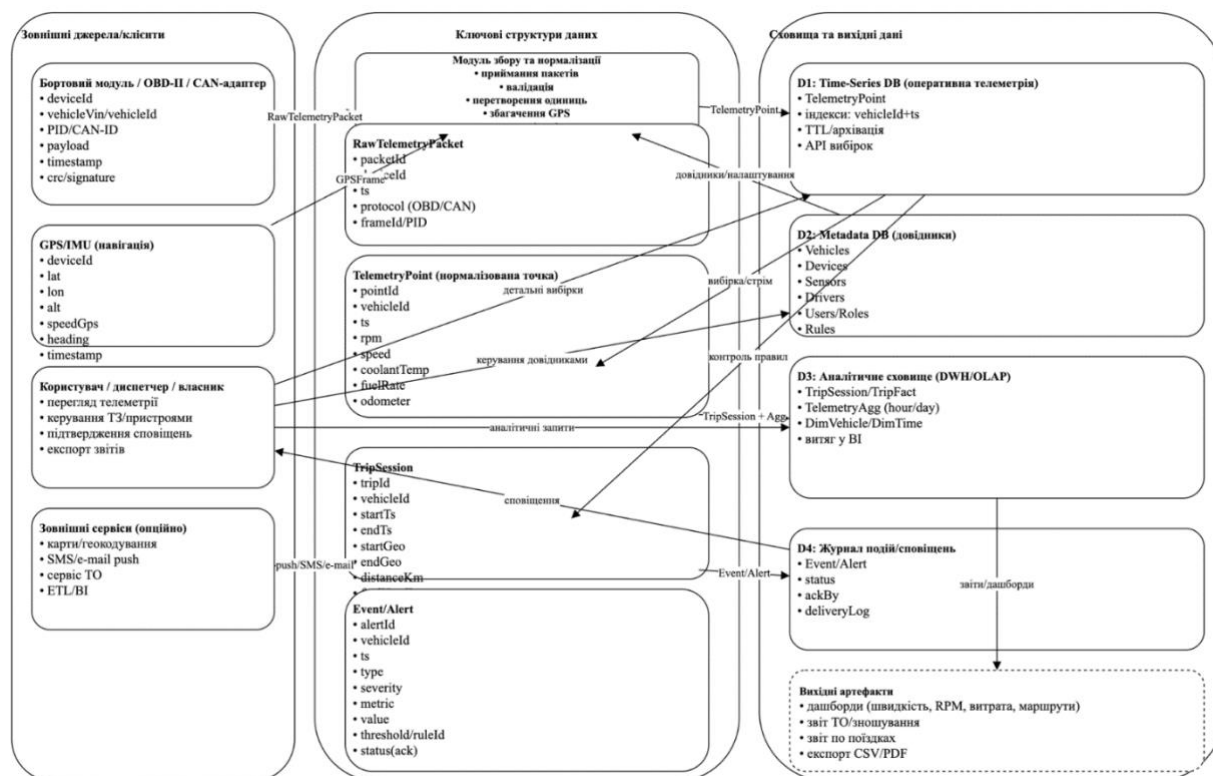


Рис. 1.1 Загальна архітектура потоків даних та компонентів інформаційної системи обліку телеметрії авто

Подальша обробка передбачає агрегацію даних за часовими інтервалами, формування сесій поїздок (TripSession), розрахунок похідних показників (пробіг, витрата пального, середня швидкість, коефіцієнти навантаження двигуна) та збереження результатів у аналітичному сховищі (DWH/OLAP). Такий підхід забезпечує можливість комплексного аналізу стану транспортного засобу, поведінки водія та ефективності експлуатації автопарку.

Важливою складовою предметної області є підсистема правил та сповіщень, що реалізує механізми контролю порогових значень параметрів, виявлення аномалій та формування подій типу Event/Alert. У разі перевищення встановлених меж або виникнення діагностичних кодів несправностей (DTC) система генерує повідомлення та передає їх користувачам через визначені канали зв'язку (push-повідомлення, SMS, електронна пошта). Журнал подій забезпечує фіксацію історії спрацювань та підтверджень.

Предметна область також передбачає інтеграцію з довідниковими та адміністративними підсистемами, що містять інформацію про транспортні засоби, пристрої, водіїв, ролі користувачів і правила моніторингу. Узгоджена взаємодія між оперативним сховищем телеметрії, метаданими та аналітичним рівнем формує цілісну інформаційно-аналітичну платформу.

Для систем обліку телеметрії характерні специфічні вимоги: підтримка високої пропускної здатності потоків даних, забезпечення масштабованості при збільшенні кількості транспортних засобів, мінімальна затримка між надходженням телеметрії та її відображенням у дашбордах, а також надійність збереження даних у разі мережевих збоїв. Крім того, система повинна забезпечувати захищеність передавання інформації та розмежування доступу відповідно до ролей користувачів.

У таблиці 1.1 наведено узагальнені характеристики предметної області та ключові вимоги до інформаційної системи обліку телеметрії авто.

Таблиця 1.1

Основні характеристики предметної області та вимоги до системи

Компонент / Характеристика	Опис
Джерела даних	OBD-II, CAN-шина, GPS/IMU, внутрішні сенсори авто
Типи задач	Збір, нормалізація, агрегація, аналіз, формування сповіщень
Обмеження	Пропускна здатність мережі, частота надходження пакетів, ресурси серверної інфраструктури
Вимоги до швидкодії	Оброблення потоків у режимі реального часу з мінімальною затримкою
Вимоги до точності	Коректна інтерпретація PID/CAN-параметрів, синхронізація часових міток
Інтеграція	Системи моніторингу автопарку, BI-сервіси, зовнішні сервіси повідомлень

Предметна область програмного забезпечення інформаційної системи обліку телеметрії авто охоплює повний цикл роботи з телеметричними потоками - від збору сирих даних з бортових пристроїв до формування аналітичних звітів і управлінських рішень. Реалізація такої системи забезпечує підвищення

прозорості експлуатації транспортних засобів, своєчасне виявлення технічних відхилень та оптимізацію витрат на обслуговування автопарку.

1.2. Аналіз існуючих рішень у сфері обліку телеметрії авто

На сучасному етапі розвитку транспортної телематики системи обліку телеметрії авто реалізуються у вигляді комплексних платформ моніторингу автопарку (Fleet Management Systems), що поєднують функції збору даних із бортових пристроїв, їх збереження, аналітичного оброблення та візуалізації. Такі рішення орієнтовані на корпоративний сегмент і забезпечують контроль технічного стану транспортних засобів, маршрутів, витрат пального та поведінки водіїв.

Існуючі рішення у сфері телеметричного моніторингу умовно можна поділити на такі основні групи: хмарні платформи керування автопарком, універсальні системи GPS-моніторингу, інтегровані екосистеми телематики з розширеною аналітикою та спеціалізовані корпоративні рішення.

Одним із найбільш поширених глобальних рішень є платформа Geotab. Архітектура системи базується на використанні власних телематичних пристроїв, що підключаються до OBD-II або CAN-шини транспортного засобу та передають дані до хмарної інфраструктури MyGeotab. Веб-інтерфейс системи з модулем аналітичних панелей наведено на рисунку 1.2.

Система забезпечує збір параметрів роботи двигуна, швидкості, витрати пального, кодування діагностичних помилок (DTC) та формування звітів щодо ефективності експлуатації автопарку. Перевагою рішення є масштабованість і розвинена аналітика, проте воно орієнтоване на використання власної екосистеми обладнання та сервісів.

Іншим відомим рішенням є платформа Wialon, що позиціонується як універсальна система GPS-моніторингу. Система підтримує широкий спектр трекерів різних виробників, надає можливість формування маршрутних звітів,

аналізу швидкісних режимів та побудови графіків руху транспортного засобу. Приклад інтерфейсу формування звітів у Wialon наведено на рисунку 1.3.

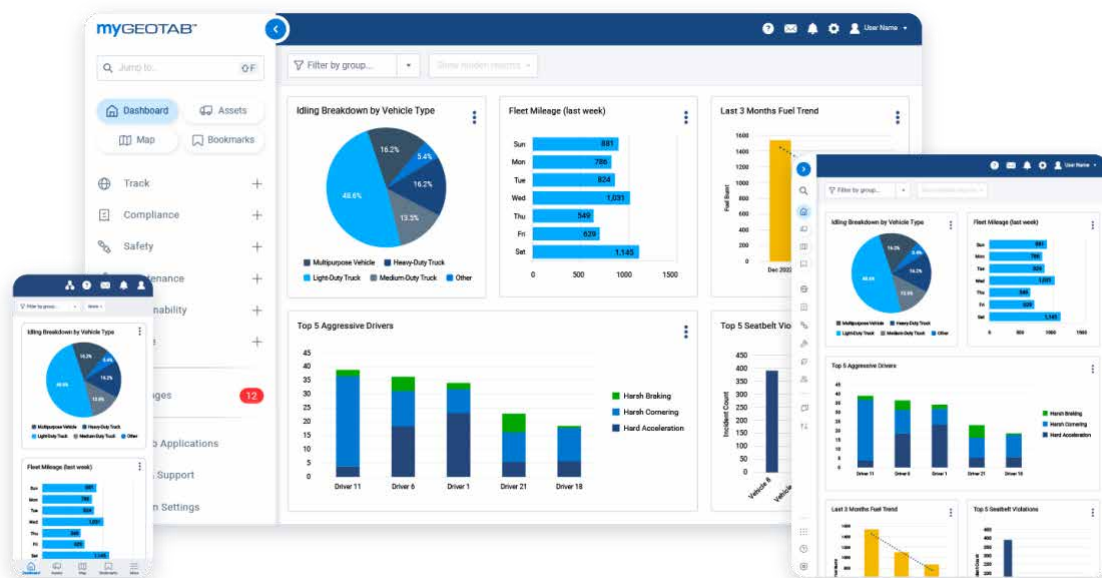


Рис. 1.2 Інтерфейс аналітичної панелі платформи Geotab (MyGeotab)

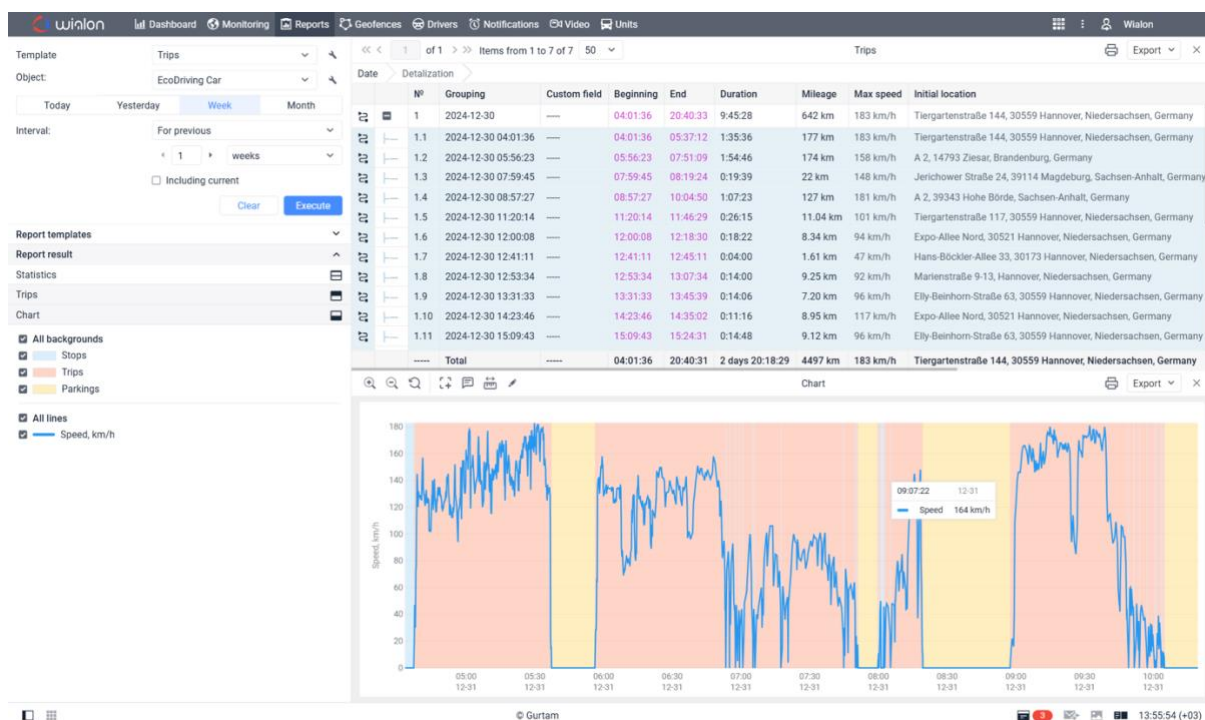


Рис. 1.3 Інтерфейс формування звітів і графіків у системі Wialon

Архітектура рішення передбачає серверний рівень оброблення даних і веб-інтерфейс із модулем побудови звітів. Перевагою є сумісність із різними

пристроями, однак аналітичні можливості обмежуються переважно базовими показниками без глибокої обробки телеметричних потоків.

Серед корпоративних рішень слід відзначити платформу Verizon Connect (Reveal). Система забезпечує комплексний моніторинг транспортних засобів, включаючи карту в реальному часі, оцінювання ефективності водіїв, аналіз стилю керування та інтеграцію з сервісами технічного обслуговування. Приклад користувацького інтерфейсу наведено на рисунку 1.4.

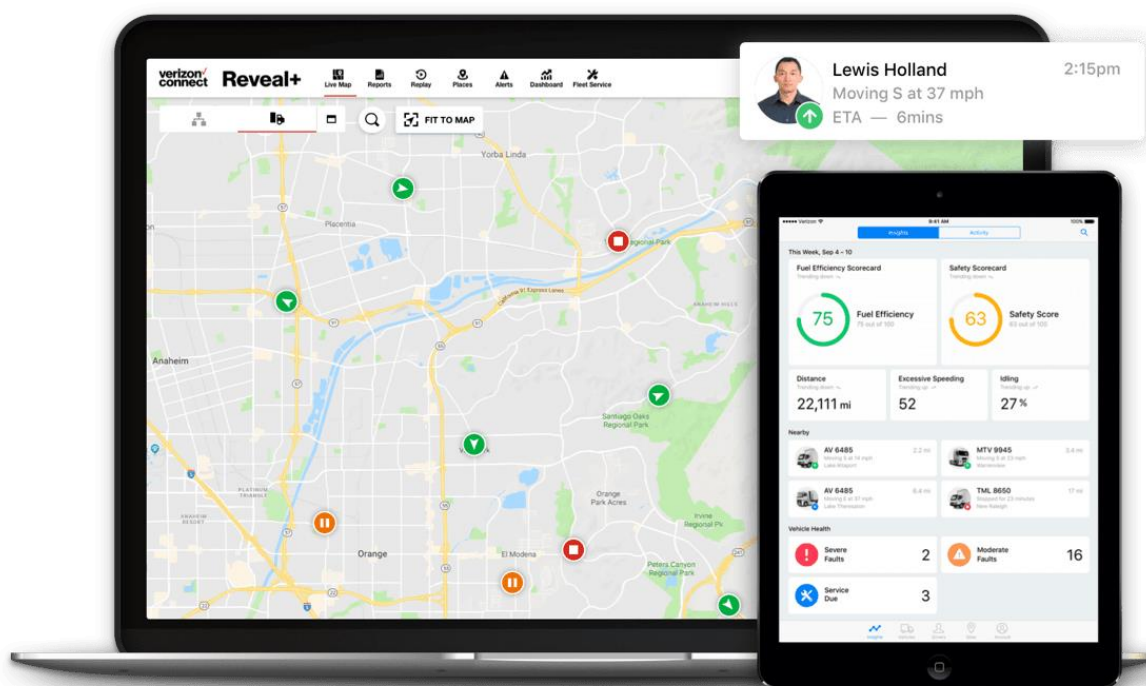


Рис. 1.4 Інтерфейс платформи Verizon Connect (Reveal)

Архітектура рішення побудована за моделлю SaaS із централізованим зберіганням даних у хмарному середовищі. Основною перевагою є глибока інтеграція з корпоративними процесами, проте система є залежною від постачальника та має обмежену гнучкість налаштування внутрішніх алгоритмів оброблення.

Платформа Samsara реалізує підхід до телеметрії як частини IoT-екосистеми. Крім стандартних параметрів транспортного засобу, система підтримує відеомоніторинг, контроль безпеки та аналітику поведінки водіїв. Приклад інформаційної панелі системи наведено на рисунку 1.5.

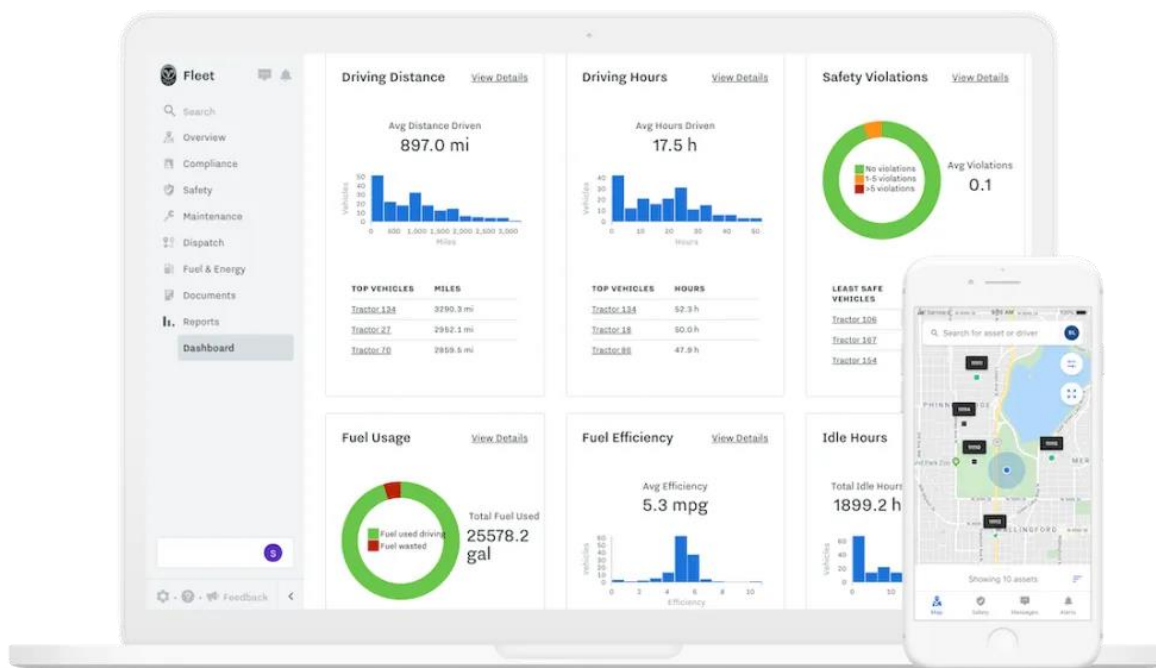


Рис. 1.5 Аналітична панель платформи Samsara

Рішення забезпечує високу масштабованість і зручність використання, однак вимагає використання фірмового обладнання та сервісної інфраструктури.

Ще одним прикладом є система Fleet Complete, що надає функціональність GPS-моніторингу, управління активами та формування аналітичних звітів. Приклад картографічного інтерфейсу наведено на рисунку 1.6.

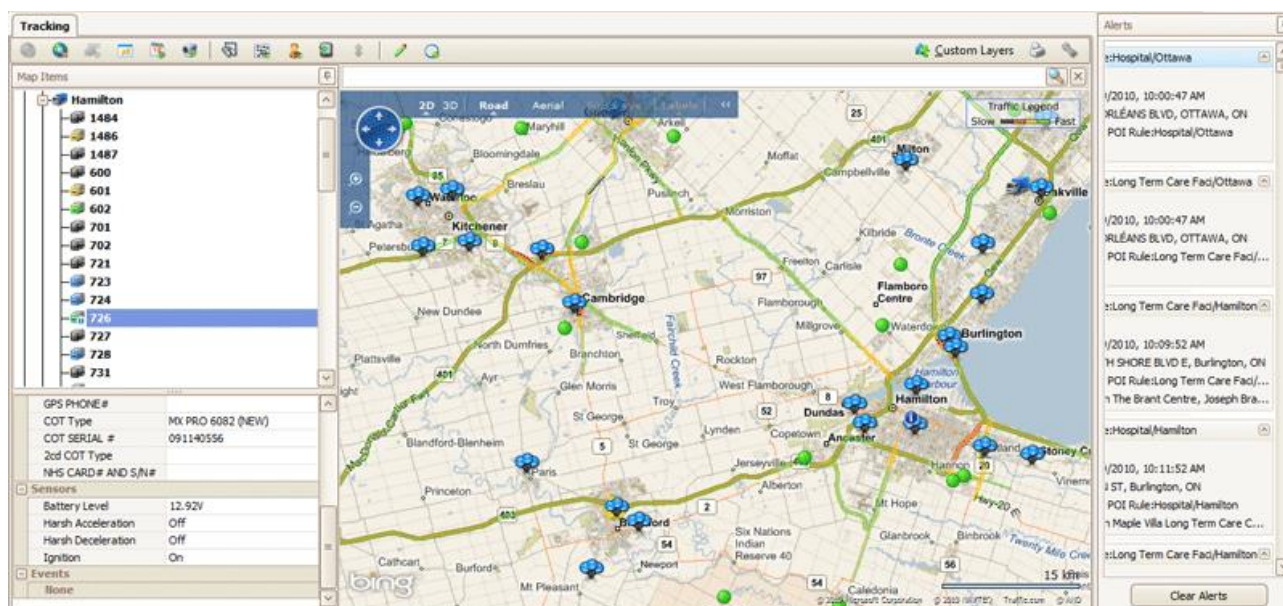


Рис. 1.6 Картографічний інтерфейс системи Fleet Complete

Платформа орієнтована на середній та малий бізнес, забезпечуючи базові функції телеметричного обліку та модуль сповіщень. Перевагою є простота впровадження, проте можливості кастомізації та глибокого аналізу потокових даних обмежені.

Порівняльну характеристику розглянутих рішень та розроблюваної системи наведено в таблиці 1.2.

Таблиця 1.2

Порівняння існуючих рішень та розроблюваної системи

Критерій	Geotab	Wialon	Verizon Connect	Samsara	Fleet Complete	Розроблювана система
Збір телеметрії з OBD/CAN	так	так	так	так	так	так
Підтримка різних пристроїв	обмежено	так	обмежено	обмежено	частково	так
Потокова обробка в реальному часі	так	частково	так	так	частково	так
Глибока аналітика та агрегація	так	обмежено	так	так	обмежено	так
Незалежність від постачальника	ні	частково	ні	ні	частково	так
Можливість кастомізації архітектури	обмежено	обмежено	низька	обмежено	низька	висока

Аналіз існуючих рішень показує, що сучасні телематичні платформи забезпечують високий рівень автоматизації збору даних і зручну візуалізацію, однак більшість із них функціонує в межах закритих екосистем із прив'язкою до конкретного обладнання або хмарної інфраструктури. Обмежена можливість адаптації алгоритмів оброблення телеметричних потоків, залежність від постачальника та недостатня гнучкість інтеграції з внутрішніми інформаційними системами підприємства знижують універсальність таких рішень.

Зазначені обмеження обґрунтовують доцільність розроблення програмного забезпечення інформаційної системи обліку телеметрії авто, яке

забезпечує незалежність від конкретного постачальника, підтримку різних джерел телеметричних даних, гнучку архітектуру потокового оброблення та інтеграцію з корпоративними аналітичними платформами.

1.3. Моделювання предметної області

Моделювання предметної області є ключовим етапом проєктування програмного забезпечення інформаційної системи обліку телеметрії авто, оскільки забезпечує формалізацію функціональних сценаріїв, взаємодії користувачів із системою та внутрішніх процесів потокового оброблення телеметричних даних. Для досягнення поставленої мети предметну область було описано за допомогою UML-моделей, які дозволяють структуровано представити поведінку системи, логіку взаємодії компонентів і послідовність виконання операцій у типових сценаріях експлуатації автотранспортних засобів.

На рисунку 1.7 наведено діаграму прецедентів, яка відображає основні функціональні можливості системи та ролі її користувачів.

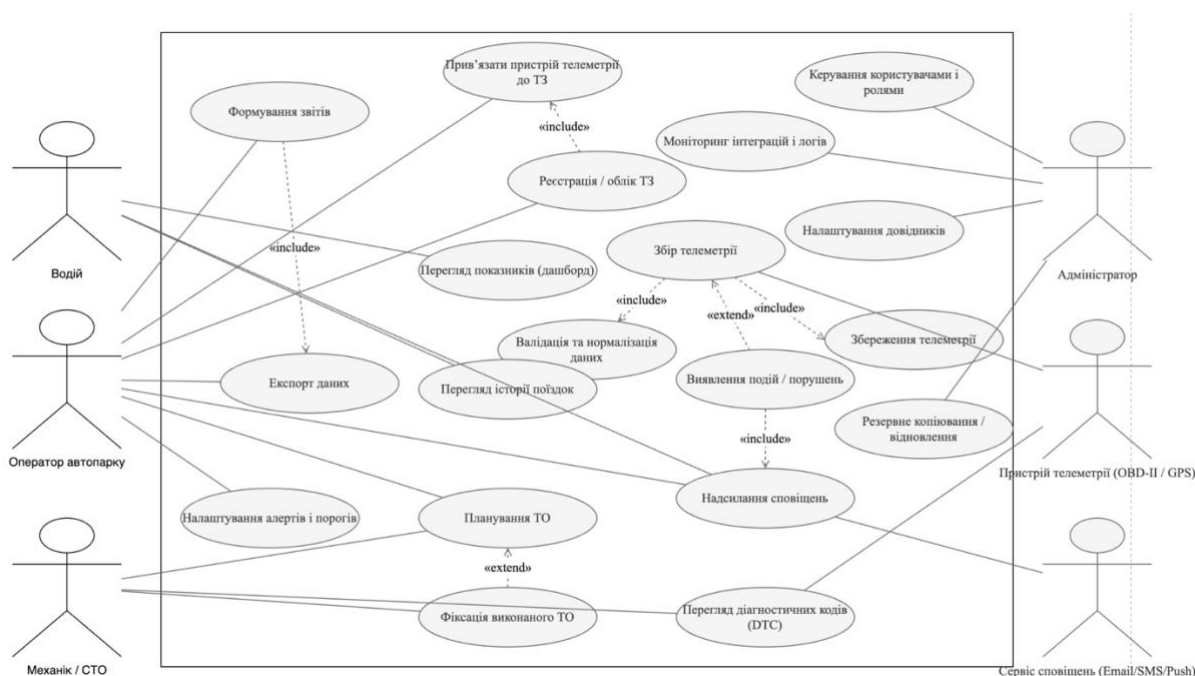


Рис. 1.7 Діаграма прецедентів інформаційної системи обліку телеметрії авто

У межах моделі виокремлено такі основні актори: водій, оператор автопарку, механік (СТО), адміністратор, а також зовнішні сутності – пристрій телеметрії (OBD-II/GPS) та сервіс сповіщень (Email/SMS/Push).

Водій взаємодіє із системою з метою перегляду показників у дашборді, перегляду історії поїздок, отримання сповіщень про перевищення порогів та формування звітів. Оператор автопарку здійснює експорт даних, налаштування алертів і порогових значень, планування технічного обслуговування, аналіз телеметричних показників автопарку. Механік відповідає за фіксацію виконаного технічного обслуговування та перегляд діагностичних кодів несправностей (DTC). Адміністратор реалізує функції керування користувачами і ролями, налаштування довідників, моніторинг інтеграцій та резервне копіювання даних.

Ключовими функціональними сценаріями системи є: збір телеметрії, валідація та нормалізація даних, збереження телеметрії, виявлення подій і порушень, надсилання сповіщень, формування звітів та планування технічного обслуговування. Застосування відношень «include» та «extend» дозволяє формально відобразити залежності між процесами, зокрема включення валідації до процесу збору телеметрії та розширення планування ТО через фіксацію виконаних робіт.

Для деталізації взаємодії між бортовим пристроєм і серверними компонентами побудовано діаграму послідовності, наведену на рисунку 1.8.

У межах моделі представлено типовий сценарій передавання телеметрії: пристрій формує пакет даних і надсилає його на API-шлюз через HTTP-запит POST /telemetry. Далі виконується перевірка токена автентифікації у сервісі автентифікації. У разі невалідного токена система повертає відповідь 401 Unauthorized, що відповідає альтернативному сценарію (alt).

За умови успішної автентифікації запит передається до сервісу телеметрії, де здійснюється перевірка схеми пакета (validateSchema) та збереження телеметричних даних у базі. Після запису формується ідентифікатор telemetryId, який використовується сервісом правил для оцінювання порогових значень. У

циклі (loop) виконується перевірка кожного правила для відповідного транспортного засобу. У разі перевищення порогів (opt) створюється запис тривоги (createAlert) та ініціюється надсилання повідомлення через сервіс сповіщень. Завершальним етапом є повернення відповіді 201 Created до пристрою.

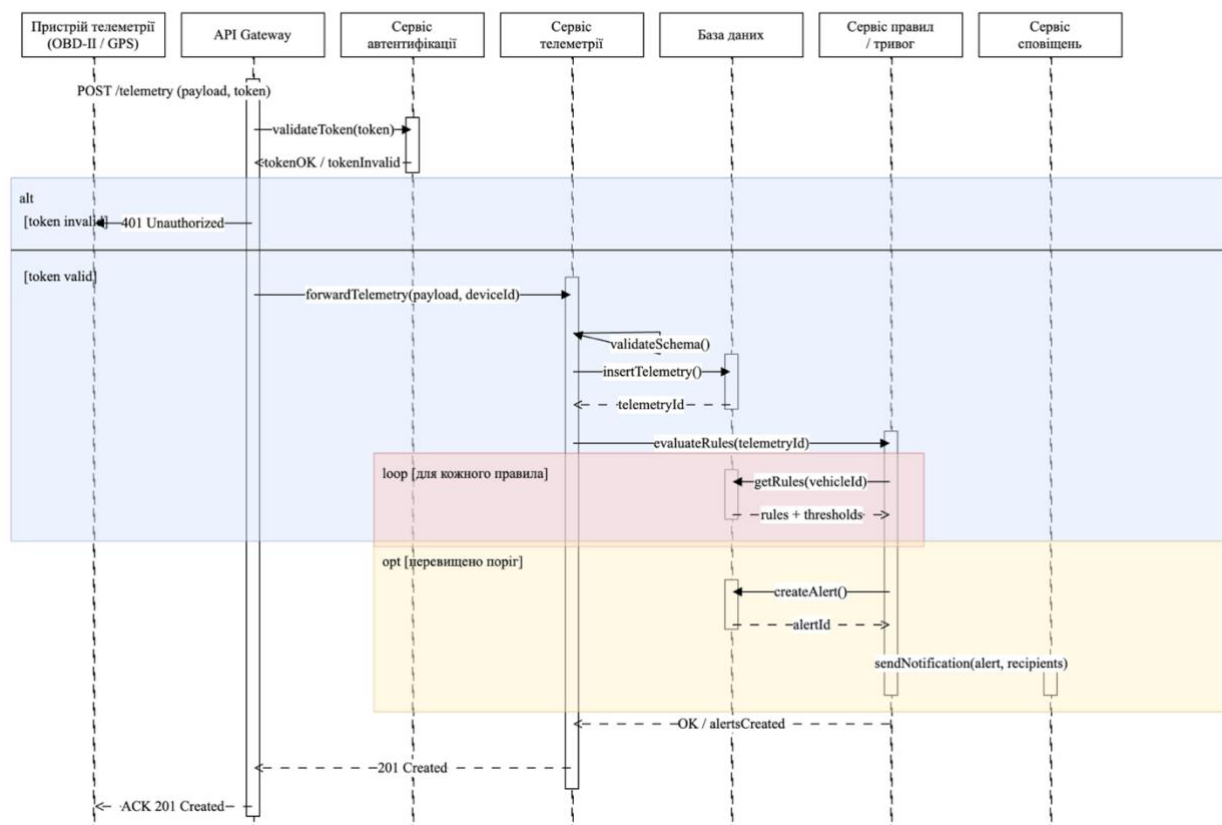


Рис. 1.8 Діаграма послідовності оброблення телеметричного пакета

Таким чином, діаграма послідовності формалізує механізм потокового оброблення телеметрії та демонструє альтернативні й умовні гілки виконання, що відповідають реальним умовам функціонування системи.

Поведенкові аспекти функціонування системи з точки зору внутрішньої логіки прийняття рішень відображено на діаграмі активності, наведеній на рисунку 1.9.

Діаграма описує повний цикл руху телеметричних даних – від зчитування показників бортовим модулем до створення запису тривоги та надсилання сповіщення. Процес розпочинається зі зчитування параметрів (GPS, швидкість, CAN/OBD), формування пакета та додавання токена й часової мітки. У разі

відсутності з'єднання пакет зберігається у буфері офлайн-черги до відновлення зв'язку.

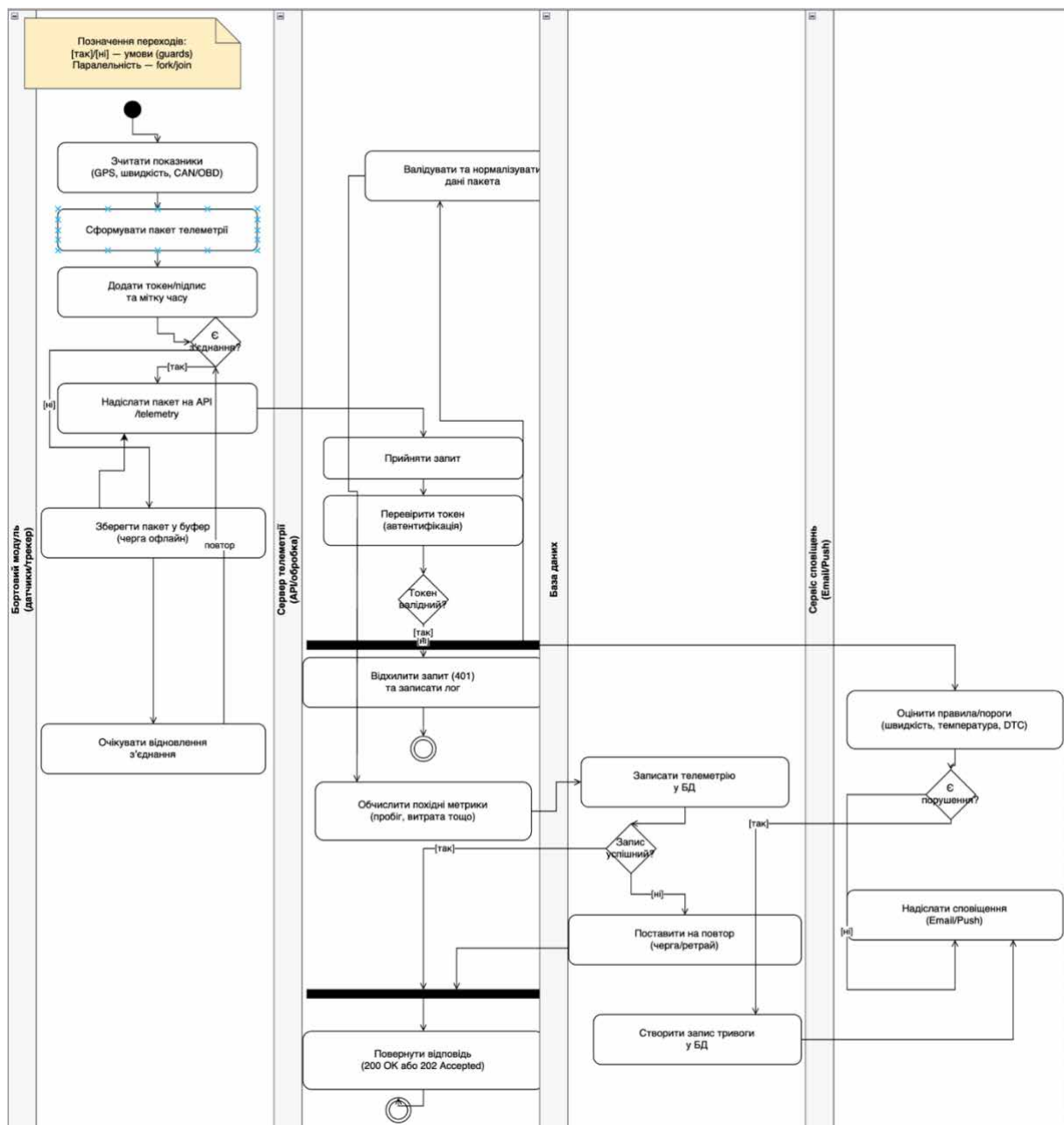


Рис. 1.9 Діаграма активності процесу збору, оброблення та аналізу телеметрії

Після надходження запиту сервер перевіряє токен автентифікації. Якщо токен невалідний, формується відповідь 401 та здійснюється запис у лог. За умови успішної перевірки виконується валідація й нормалізація даних, обчислення похідних метрик (пробіг, витрата пального), запис телеметрії у базу даних та перевірка правил і порогів (швидкість, температура, DTC). У випадку

виявлення порушення формується запис тривоги та надсилається повідомлення відповідним отримувачам.

Таким чином, діаграма активності дозволяє простежити алгоритмічну логіку функціонування системи, включаючи умовні переходи, паралельні процеси та механізми повторної спроби запису (retry).

Узагальнюючи результати моделювання, можна стверджувати, що використаний комплекс UML-діаграм забезпечує цілісне формальне представлення предметної області інформаційної системи обліку телеметрії авто. Побудовані моделі дозволяють обґрунтовано перейти до етапів проєктування архітектури та програмної реалізації, оскільки чітко визначають ролі користувачів, структуру інформаційних потоків, механізми оброблення телеметрії та логіку формування подій і сповіщень.

1.4. Аналіз вимог розроблюваної системи

Розроблення програмного забезпечення інформаційної системи обліку телеметрії авто потребує попереднього формування та комплексного аналізу вимог, які визначають її функціональне призначення, експлуатаційні характеристики, технічні обмеження та рівень захищеності даних. Коректно сформульовані вимоги забезпечують узгодженість між особливостями предметної області, сценаріями використання системи та можливостями програмної реалізації, а також формують основу для подальшого проєктування архітектури й вибору технологічного стеку.

Передусім визначаються функціональні вимоги, які описують перелік задач, що повинна виконувати система в процесі збору, збереження та аналізу телеметричних даних транспортних засобів. Вони охоплюють повний цикл оброблення телеметрії – від приймання пакета даних до формування аналітичних звітів і сповіщень. Узагальнений перелік функціональних вимог наведено в таблиці 1.3.

Таблиця 1.3

Функціональні вимоги до інформаційної системи обліку телеметрії авто

№	Вимога
1	Приймання телеметричних даних від бортових пристроїв (OBD-II, CAN, GPS)
2	Валідація та нормалізація телеметричних пакетів
3	Збереження телеметричних даних у базі часових рядів
4	Формування сесій поїздок (TripSession)
5	Обчислення похідних метрик (пробіг, витрата пального, середня швидкість)
6	Виявлення подій та порушень за заданими правилами
7	Надсилання сповіщень користувачам
8	Формування аналітичних звітів і дашбордів
9	Управління користувачами, ролями та довідниками
10	Підтримка планування та фіксації технічного обслуговування

Як видно з таблиці 1.3, функціональні вимоги охоплюють як оброблення поточкових телеметричних даних, так і аналітичну та адміністративну складову системи.

Окрім функціональних можливостей, визначено нефункціональні вимоги, що характеризують якість функціонування системи в умовах реальної експлуатації автопарку. Вони визначають продуктивність, масштабованість, доступність та надійність роботи. Основні нефункціональні вимоги наведено в таблиці 1.4.

Таблиця 1.4

Нефункціональні вимоги до системи

№	Вимога
1	Оброблення телеметричних потоків у режимі, близькому до реального часу

2	Висока доступність сервісів (24/7)
3	Масштабованість при зростанні кількості транспортних засобів
4	Відмовостійкість та підтримка механізмів повторної обробки (retry)
5	Мінімальна затримка між отриманням телеметрії та її відображенням у дашборді
6	Можливість інтеграції з зовнішніми BI- та ERP-системами

З таблиці 1.4 випливає, що нефункціональні вимоги спрямовані на забезпечення стабільної роботи системи в умовах високої інтенсивності надходження даних та динамічного розширення автопарку.

Технічні вимоги визначають апаратно-програмні обмеження, протоколи взаємодії та технологічні засоби реалізації системи. Вони формують основу для вибору архітектурних рішень і програмних інструментів. Перелік основних технічних вимог наведено в таблиці 1.5.

Таблиця 1.5

Технічні вимоги до інформаційної системи обліку телеметрії авто

№	Вимога
1	Підтримка REST API для приймання телеметричних даних
2	Використання захищеного протоколу HTTPS
3	Підтримка автентифікації пристроїв (token/mTLS)
4	Використання бази даних часових рядів для збереження телеметрії
5	Наявність механізмів резервного копіювання та відновлення даних
6	Можливість розгортання у хмарному або локальному середовищі
7	Підтримка контейнеризації та мікросервісної архітектури

Оскільки система працює з даними, що можуть містити конфіденційну інформацію (геолокацію, поведінкові характеристики водія, технічні параметри транспортного засобу), особливу увагу приділено вимогам до інформаційної безпеки. Основні вимоги до безпеки наведено в таблиці 1.6.

Таблиця 1.6

Вимоги до інформаційної безпеки системи

№	Вимога
1	Обов'язкова автентифікація та авторизація користувачів і пристроїв
2	Шифрування даних під час передавання та зберігання
3	Розмежування прав доступу відповідно до ролей
4	Захист API від несанкціонованих запитів (rate limiting, логування)
5	Забезпечення цілісності та незмінності журналів подій
6	Реєстрація та моніторинг інцидентів безпеки

На підставі таблиці 1.6 можна зробити висновок, що безпекові вимоги охоплюють як програмні, так і інфраструктурні аспекти функціонування системи та є критично важливими для її практичного застосування в корпоративному середовищі.

Проведений аналіз вимог дозволив сформулювати цілісне уявлення про характеристики розроблюваної інформаційної системи обліку телеметрії авто. Сформульовані функціональні, нефункціональні, технічні та безпекові вимоги створюють основу для подальшого проєктування архітектури, вибору технологій реалізації та розроблення програмного забезпечення, орієнтованого на ефективну й безпечну експлуатацію автопарку.

1.5. Постановка завдання

У межах кваліфікаційної роботи необхідно розв'язати задачу проєктування та програмної реалізації інформаційної системи обліку телеметрії авто шляхом створення програмної моделі, що відтворює логіку функціонування реальної системи моніторингу транспортних засобів. Розроблювана система повинна моделювати основні процеси збору, передавання, збереження, аналізу та візуалізації телеметричних даних, а також забезпечувати контроль подій, формування сповіщень і ведення журналу операцій.

Постановка завдання передбачає розроблення програмної моделі системи, яка імітує взаємодію бортових телеметричних пристроїв (OBD-II, CAN, GPS), серверних компонентів оброблення даних, підсистеми правил і тривог, бази даних часових рядів та клієнтських інтерфейсів користувачів. Такий підхід дозволяє дослідити принципи функціонування системи моніторингу автопарку в умовах потокового надходження даних без необхідності використання фізичних транспортних засобів на етапі проєктування.

Вхідними даними розроблюваної системи є телеметричні пакети, що надходять від бортових пристроїв транспортних засобів. До складу таких даних належать: ідентифікатор пристрою або транспортного засобу, часові мітки, показники швидкості, обертів двигуна (RPM), температури охолоджувальної рідини, витрати пального, координати GPS, діагностичні коди несправностей (DTC) та інші параметри, що передаються через OBD-II або CAN-шину. Крім того, до вхідних даних належать службові атрибути автентифікації (токени доступу), а також адміністративні запити користувачів щодо формування звітів, налаштування порогових значень і керування довідниками.

Вихідними даними системи є результати оброблення телеметрії та прийняття рішень на її основі. До них належать: збережені телеметричні записи у базі даних, сформовані сесії поїздок, розраховані агреговані показники (пробіг, середня швидкість, витрата пального), сформовані події та тривоги при перевищенні порогових значень, повідомлення користувачам через Email/SMS/Push, а також аналітичні звіти й дані для дашбордів. Усі події та операції повинні фіксуватися у журналі для подальшого аналізу та аудиту.

У процесі розв'язання поставленої задачі необхідно реалізувати такі основні функції програмної моделі системи:

- імітацію надходження телеметричних пакетів від бортових пристроїв;
- автентифікацію та авторизацію пристроїв і користувачів;
- валідацію та нормалізацію телеметричних даних;
- збереження даних у базі часових рядів;
- обчислення похідних метрик та формування сесій поїздок;

- перевірку телеметрії відповідно до налаштованих правил і порогів;
- створення записів тривог і надсилання сповіщень;
- формування звітів і візуалізацію показників у веб-інтерфейсі;
- ведення журналу подій та забезпечення механізмів резервного копіювання.

Програмна реалізація повинна мати модульну архітектуру з розділенням на окремі сервіси (API-шлюз, сервіс телеметрії, сервіс правил, сервіс сповіщень, підсистема збереження даних), що забезпечує масштабованість, розширюваність і можливість інтеграції з зовнішніми системами управління автопарком або ВІ-платформами. Передбачається підтримка роботи в умовах потокового навантаження та забезпечення оброблення даних у режимі, близькому до реального часу.

Постановка завдання полягає у створенні програмної моделі інформаційної системи обліку телеметрії авто, яка відтворює повний цикл роботи з телеметричними даними – від їх отримання до формування аналітичних результатів і сповіщень, забезпечує контроль доступу та журналювання операцій і слугує основою для подальшого проектування та практичної реалізації повнофункціональної системи моніторингу транспортних засобів.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Проектування інформаційного забезпечення програмного забезпечення інформаційної системи обліку телеметрії авто починається з побудови логічної моделі даних. Логічна модель даних дає змогу формалізувати структуру інформації, яка використовується системою, визначити основні сутності предметної області, їх атрибути та взаємозв'язки між ними. Для наочного подання структури даних використано ER-діаграму, що наведена на рис. 2.1.

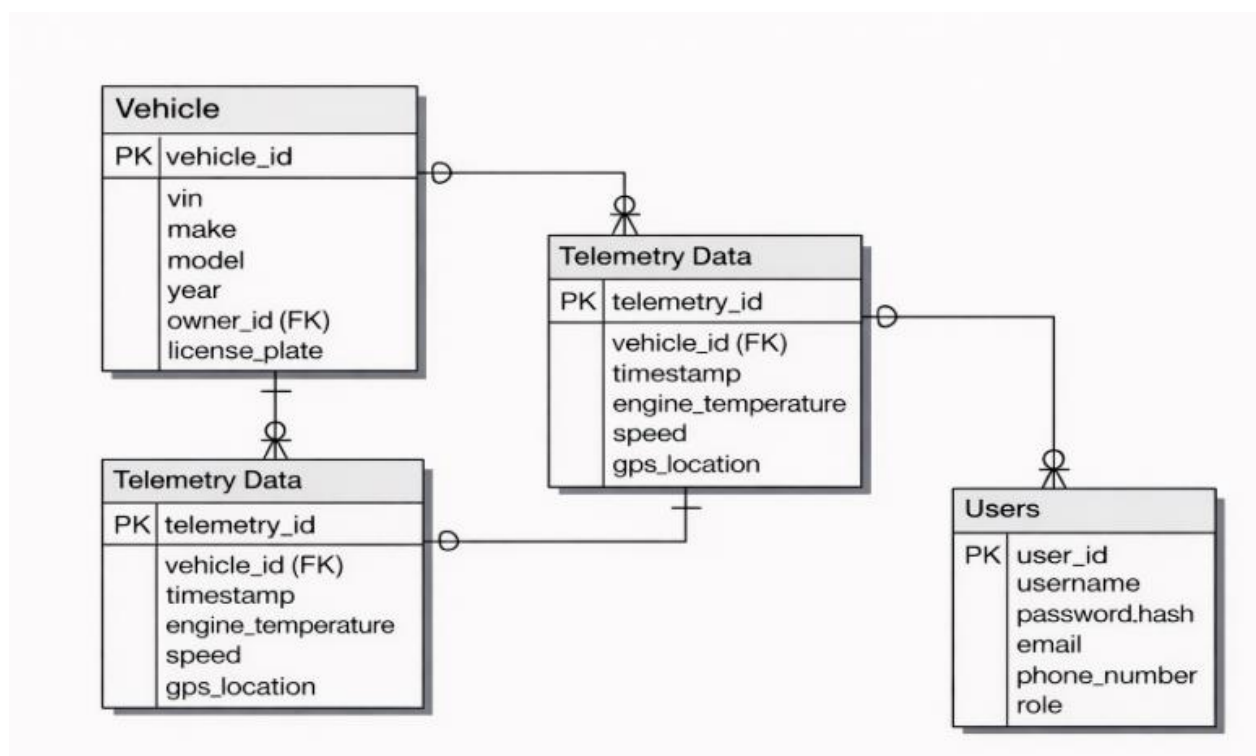


Рис. 2.1 ER-діаграма логічної моделі даних інформаційної системи обліку телеметрії авто

Побудована ER-діаграма відображає основні інформаційні об'єкти системи, які забезпечують збереження відомостей про користувачів, транспортні засоби та телеметричні дані, що надходять від бортових пристроїв. У межах

логічної моделі виокремлено три базові сутності: Users, Vehicle та Telemetry Data.

Сутність Users призначена для збереження даних про користувачів системи. До її складу входять такі атрибути: `user_id` - первинний ключ, `username` - ім'я користувача в системі, `password_hash` - хеш пароля, `email` - адреса електронної пошти, `phone_number` - номер телефону, `role` - роль користувача. Наявність ролі в структурі сутності забезпечує можливість реалізації розмежування прав доступу до функціональних можливостей системи. У межах розроблюваної системи до користувачів можуть належати власники транспортних засобів, оператори автопарку, технічні спеціалісти або адміністратори.

Сутність Vehicle описує транспортний засіб як об'єкт моніторингу. Вона містить атрибути: `vehicle_id` — первинний ключ, `vin` — унікальний ідентифікаційний номер транспортного засобу, `make` — марка, `model` — модель, `year` — рік випуску, `owner_id` — зовнішній ключ на користувача-власника, `license_plate` — державний номерний знак. Така структура дає змогу зберігати основну облікову інформацію про автомобілі, а також пов'язувати кожний транспортний засіб з конкретним користувачем системи.

Центральною сутністю логічної моделі є Telemetry Data, яка акумулює телеметричну інформацію, отриману від бортових пристроїв автомобіля. До її атрибутів належать: `telemetry_id` - первинний ключ, `vehicle_id` - зовнішній ключ на транспортний засіб, `timestamp` - час фіксації телеметричного запису, `engine_temperature` - температура двигуна, `speed` - швидкість руху, `gps_location` - координати місцезнаходження транспортного засобу. Саме ця сутність забезпечує накопичення динамічних даних, які в подальшому використовуються для аналізу технічного стану автомобіля, контролю режимів його експлуатації, формування звітів і візуалізації маршрутної інформації.

Між сутностями логічної моделі встановлено відповідні зв'язки. Між сутностями Users і Vehicle реалізовано зв'язок типу один-до-багатьох: один користувач може бути власником одного або кількох транспортних засобів, тоді

як кожний транспортний засіб у межах моделі закріплюється за одним власником. Такий зв'язок реалізується за допомогою зовнішнього ключа `owner_id` у сутності `Vehicle`.

Між сутностями `Vehicle` і `Telemetry Data` також реалізовано зв'язок типу один-до-багатьох. Один транспортний засіб у процесі експлуатації генерує значну кількість телеметричних записів, тоді як кожний окремий телеметричний запис належить лише одному автомобілю. Реалізація цього зв'язку здійснюється за допомогою зовнішнього ключа `vehicle_id` у сутності `Telemetry Data`. Саме цей взаємозв'язок є основою для накопичення історії телеметрії та подальшого аналізу змін параметрів у часі.

Взаємодія користувачів із телеметричними даними в системі реалізується опосередковано через транспортні засоби. Це означає, що користувач отримує доступ до телеметрії тих автомобілів, які з ним пов'язані на рівні облікових даних або прав доступу. Такий підхід є логічно обґрунтованим, оскільки дозволяє уникнути надлишкового дублювання зв'язків у моделі та забезпечує цілісність інформаційної бази.

Запропонована логічна модель даних відповідає основним вимогам до реляційного проектування. Кожна сутність має первинний ключ, атрибути є атомарними, а зв'язки між сутностями визначені через зовнішні ключі. Модель не містить явного дублювання облікових даних користувачів і транспортних засобів, а телеметричні параметри винесені в окрему сутність, що відповідає логіці предметної області. Це дає підстави стверджувати, що логічна модель є придатною для подальшого переходу до фізичної моделі бази даних.

Отже, розроблена ER-діаграма відображає основні інформаційні об'єкти програмного забезпечення інформаційної системи обліку телеметрії авто та зв'язки між ними. Вона формує концептуальну основу для побудови структури бази даних, реалізації запитів, організації збереження телеметричних потоків і забезпечення подальшої аналітичної обробки даних. На наступному етапі проектування на основі цієї логічної моделі буде виконано розроблення фізичної структури бази даних та SQL-опису її об'єктів.

2.2 Діаграма класів і кооперації

Після побудови логічної моделі даних доцільно перейти до об'єктно-орієнтованого подання програмного забезпечення інформаційної системи обліку телеметрії авто. Для цього розроблено діаграму класів, яка відображає основні програмні абстракції предметної області, їх атрибути, операції та взаємозв'язки. Така модель є основою для подальшого проєктування програмних модулів, визначення бізнес-логіки системи та розподілу відповідальності між об'єктами.

На рис. 2.2 наведено діаграму класів інформаційної системи обліку телеметрії авто.

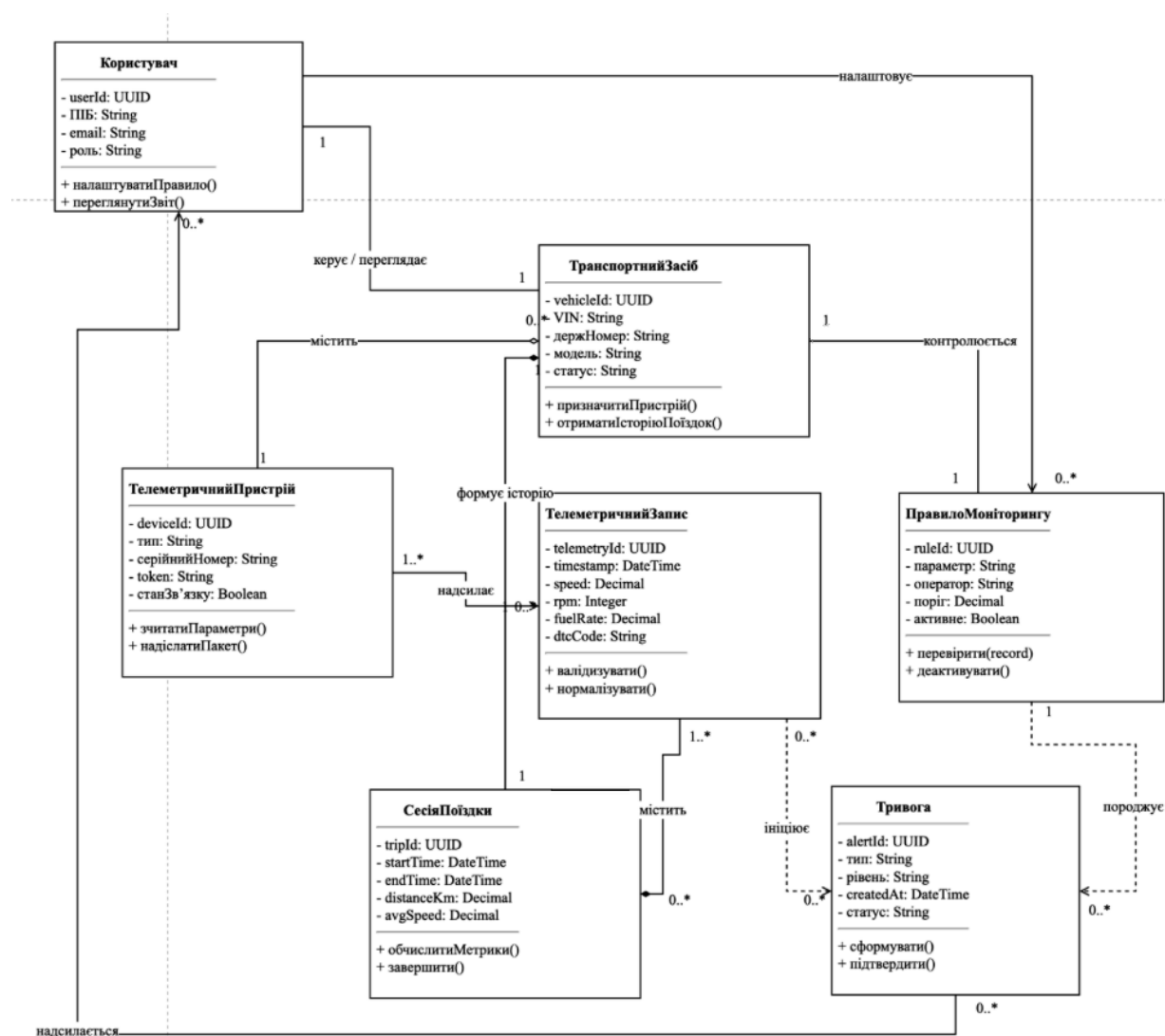


Рис. 2.2 Діаграма класів інформаційної системи обліку телеметрії авто

Діаграма класів відображає структуру основних об'єктів системи, які беруть участь у процесах керування транспортними засобами, збору телеметричних даних, формування сесій поїздок, перевірки правил моніторингу та створення тривоги. Центральними класами моделі є ТранспортнийЗасіб, ТелеметричнийПристрій, ТелеметричнийЗапис, СесіяПоїздки, ПравилоМоніторингу, Тривога та Користувач.

Клас Користувач описує суб'єкта, який працює із системою. Він містить ідентифікатор користувача, ПІБ, електронну пошту та роль. Через методи налаштуватиПравило() і переглянутиЗвіт() користувач може змінювати правила контролю параметрів і отримувати аналітичну інформацію про стан транспортного засобу.

Клас ТранспортнийЗасіб є основною обліковою одиницею системи. Він містить ідентифікатор автомобіля, VIN-код, державний номер, модель і статус. Методи призначитиПристрій() та отриматиІсторіюПоїздок() забезпечують прив'язку телеметричного пристрою до автомобіля та доступ до історії його експлуатації.

Клас ТелеметричнийПристрій відповідає за джерело надходження даних. Він містить тип пристрою, серійний номер, токен доступу та стан зв'язку. Операції зчитатиПараметри() і надіслатиПакет() відображають логіку отримання параметрів із бортових систем автомобіля та передавання сформованого пакета на серверну частину системи.

Клас ТелеметричнийЗапис описує окремий запис телеметрії, отриманий від пристрою. До його атрибутів належать дата й час фіксації, швидкість, оберти двигуна, витрата пального та діагностичний код несправності. Методи валідувати() і нормалізувати() використовуються для перевірки коректності отриманих даних і приведення їх до єдиного формату збереження.

Клас СесіяПоїздки призначений для групування телеметричних записів у межах окремої поїздки. Він містить час початку й завершення поїздки, пройдену відстань і середню швидкість. Методи обчислитиМетрики() та завершити()

забезпечують розрахунок агрегованих показників і фіксацію завершення поїздки.

Клас `ПравилоМоніторингу` описує умови контролю телеметричних параметрів. Він містить параметр, оператор порівняння, порогове значення та ознаку активності. Метод `перевірити(record)` використовується для аналізу телеметричного запису, а метод `деактивувати()` - для вимкнення правила без його фізичного видалення з системи.

Клас `Тривога` використовується для збереження інформації про зафіксовані порушення або критичні події. Він містить тип тривоги, рівень важливості, дату створення та статус. Методи `сформувати()` і `підтвердити()` забезпечують створення повідомлення про подію та фіксацію факту його оброблення користувачем.

Для узагальнення призначення класів у табл. 2.1 наведено їх коротку характеристику.

Таблиця 2.1

Характеристика основних класів системи

Клас	Призначення	Основні операції
Користувач	Зберігає дані про користувача та його роль у системі	<code>налаштуватиПравило()</code> , <code>переглянутиЗвіт()</code>
ТранспортнийЗасіб	Описує автомобіль як об'єкт моніторингу	<code>призначитиПристрій()</code> , <code>отриматиІсторіюПоїздок()</code>
ТелеметричнийПристрій	Формує та передає телеметричні пакети	<code>зчитатиПараметри()</code> , <code>надіслатиПакет()</code>
ТелеметричнийЗапис	Зберігає окремий набір телеметричних параметрів	<code>валідувати()</code> , <code>нормалізувати()</code>
СесіяПоїздки	Групує телеметричні записи в межах однієї поїздки	<code>обчислитиМетрики()</code> , <code>завершити()</code>

Закінчення Таблиці 2.1

ПравилоМоніторингу	Визначає умови контролю параметрів автомобіля	перевірити(record), деактивувати()
Тривога	Фіксує критичні події та порушення	сформувати(), підтвердити()

Зв'язки між класами відповідають логіці предметної області. Один користувач може керувати або переглядати кілька транспортних засобів, тоді як кожний транспортний засіб контролюється в межах заданих правил моніторингу. До одного транспортного засобу може бути прив'язаний телеметричний пристрій, який формує потік телеметричних записів. Записи об'єднуються в сесії поїздок, а також перевіряються правилами моніторингу. У разі виявлення порушення правило породжує одну або кілька тривог.

Для деталізації взаємодії між класами побудовано три кооперації, які відображають основні сценарії роботи системи. Кооперації дозволяють показати не лише статичну структуру класів, а й порядок їх взаємодії під час виконання прикладних функцій.

Перша кооперація відображає сценарій прив'язки телеметричного пристрою до транспортного засобу. Її наведено на рис. 2.3.

У цьому сценарії користувач спочатку обирає транспортний засіб, після чого об'єкт ТранспортнийЗасіб виконує операцію призначитиПристрій(device). Далі телеметричний пристрій прив'язується до автомобіля, а результат операції підтверджується користувачеві. Така кооперація описує початкове налаштування системи перед збором телеметрії.

Друга кооперація описує процес отримання та оброблення телеметричного пакета. Її наведено на рис. 2.4.

У межах цієї кооперації телеметричний пристрій надсилає пакет даних, на основі якого створюється об'єкт ТелеметричнийЗапис. Далі запис проходить валідацію та нормалізацію, після чого додається до відповідної сесії поїздки. Клас СесіяПоїздки виконує розрахунок основних метрик, зокрема середньої

швидкості, пробігу та інших агрегованих показників. Після цього оновлюється історія поїздок транспортного засобу.

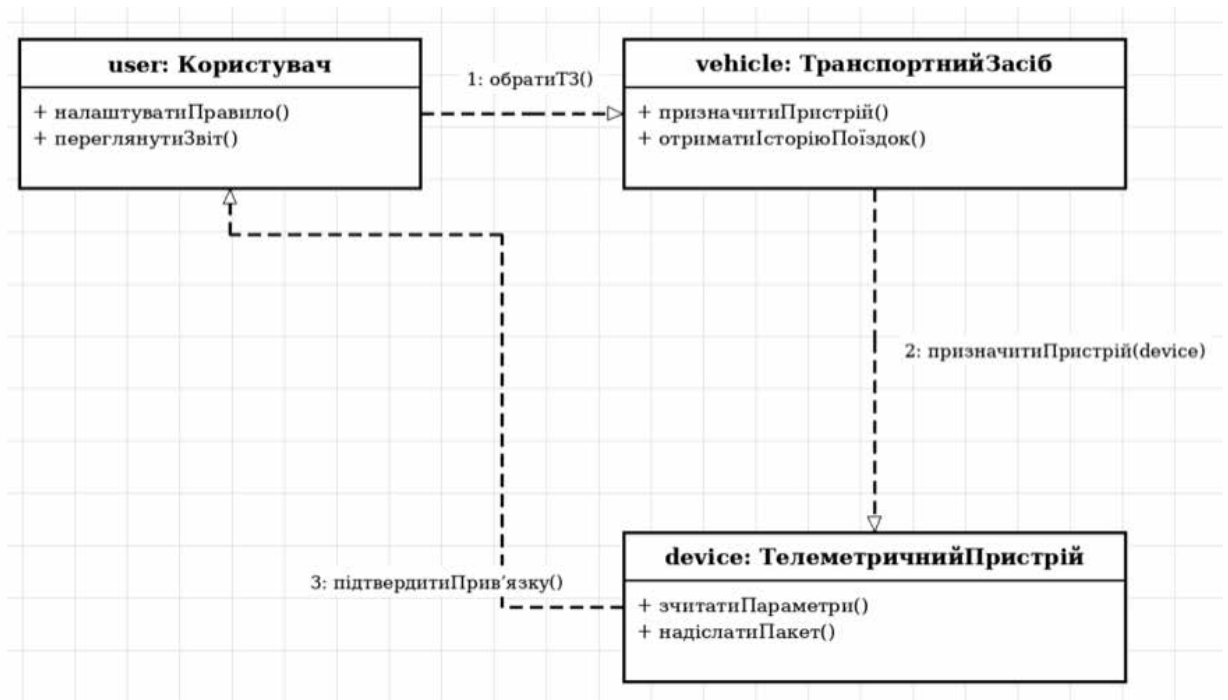


Рис. 2.3 Кооперація класів під час прив'язки телеметричного пристрою до транспортного засобу

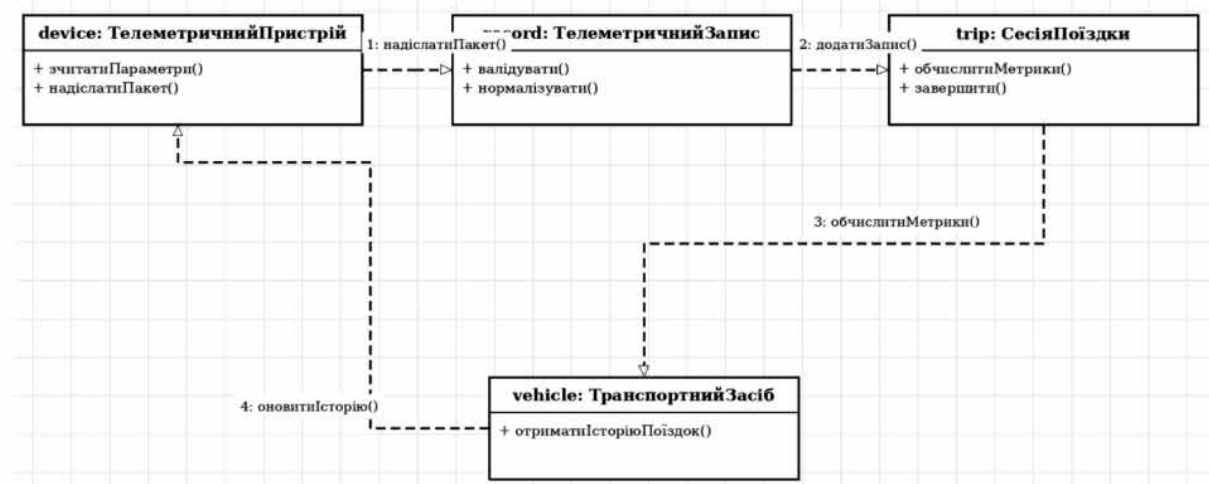


Рис. 2.4 Кооперація класів під час оброблення телеметричного пакета

Третя кооперація відображає сценарій перевірки телеметричного запису за правилами моніторингу та формування тривоги. Її наведено на рис. 2.5.

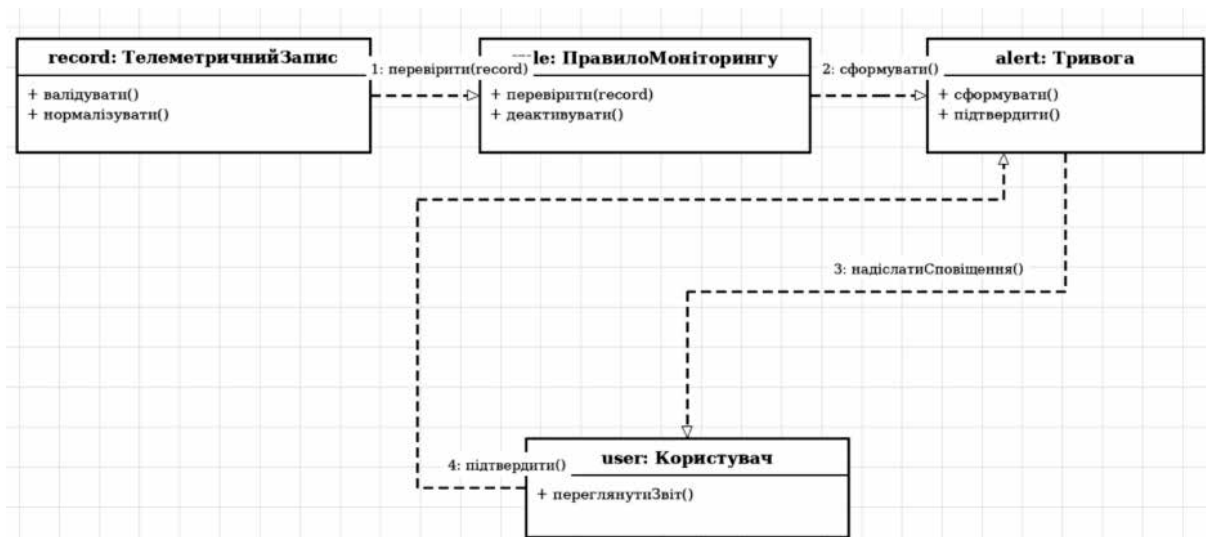


Рис. 2.5 Кооперація класів під час формування тривоги за результатами моніторингу

У цьому сценарії телеметричний запис передається до класу ПравилоМоніторингу, де виконується перевірка параметрів відповідно до встановленого порогу. Якщо умову порушено, створюється об'єкт Тривога. Після формування тривоги користувач отримує сповіщення та може підтвердити його оброблення. Такий механізм забезпечує оперативне реагування на перевищення швидкості, критичну температуру двигуна, появу DTC-кодів або інші відхилення.

Отже, діаграма класів і побудовані кооперації формалізують структуру програмного забезпечення інформаційної системи обліку телеметрії авто. Діаграма класів визначає основні об'єкти системи, їхні атрибути, методи та зв'язки, а кооперації демонструють логіку взаємодії цих об'єктів у ключових сценаріях: прив'язка пристрою до автомобіля, оброблення телеметричного пакета та формування тривоги. Побудовані моделі є основою для подальшого проєктування пакетної, компонентної та алгоритмічної структури програмного забезпечення.

2.3 Діаграма компонентів

Компонентна модель програмного забезпечення використовується для подання загальної архітектури інформаційної системи обліку телеметрії авто на рівні функціональних модулів, сервісів, баз даних і зовнішніх інтеграцій. На відміну від діаграми класів, яка описує внутрішню об'єктну структуру системи, діаграма компонентів відображає фізично або логічно відокремлені частини програмного забезпечення та способи їх взаємодії через інтерфейси.

Діаграму компонентів інформаційної системи обліку телеметрії авто наведено на рис. 2.6.

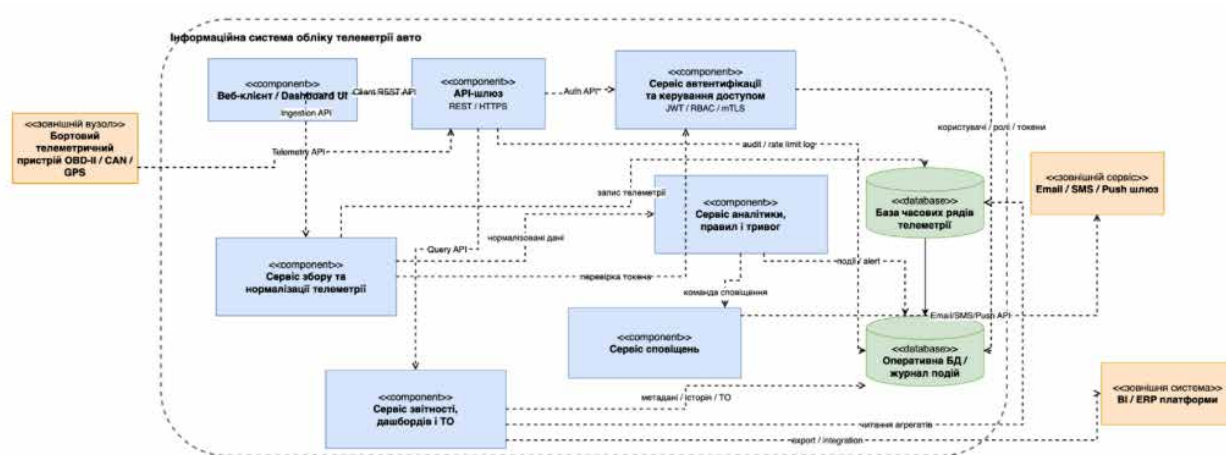


Рис. 2.6 Діаграма компонентів інформаційної системи обліку телеметрії авто

На діаграмі компонентів відображено структуру системи, побудовану за модульним принципом. Центральною частиною архітектури є межа інформаційної системи обліку телеметрії авто, у якій розміщено основні програмні компоненти: веб-клієнт, API-шлюз, сервіс автентифікації та керування доступом, сервіс збору й нормалізації телеметрії, сервіс аналітики, правил і тривог, сервіс звітності, дашбордів і технічного обслуговування, сервіс сповіщень, базу часових рядів телеметрії та оперативну базу даних журналу подій.

Зовнішнім джерелом даних є бортові телеметричні пристрої OBD-II, CAN та GPS, які передають телеметричні пакети до системи через захищений API. Приймання запитів виконує API-шлюз, що забезпечує маршрутизацію

HTTP/REST-запитів до відповідних внутрішніх сервісів. Доступ користувачів і пристроїв контролюється сервісом автентифікації та керування доступом, який реалізує механізми JWT, RBAC і mTLS. Це дозволяє перевіряти токени, обмежувати права доступу та вести аудит дій.

Сервіс збору та нормалізації телеметрії відповідає за приймання сирих пакетів, перевірку їх структури, приведення параметрів до єдиного формату та передавання нормалізованих даних у подальшу обробку. Телеметричні записи зберігаються у базі часових рядів, що є оптимальним рішенням для накопичення великої кількості значень, прив'язаних до часових міток. Оперативна база даних і журнал подій використовуються для збереження службових записів, інформації про тривоги, дії користувачів, історію сповіщень та результатів перевірок.

Сервіс аналітики, правил і тривог виконує перевірку телеметричних параметрів відповідно до налаштованих порогових значень. У разі виявлення перевищення швидкості, критичної температури двигуна, появи діагностичного коду несправності або іншого порушення сервіс формує подію тривоги та передає команду до сервісу сповіщень. Сервіс сповіщень забезпечує надсилання повідомлень користувачам через зовнішні канали Email, SMS або Push.

Сервіс звітності, дашбордів і технічного обслуговування використовує агреговані телеметричні дані, метадані про транспортні засоби та історію подій для формування аналітичних панелей, звітів і рекомендацій щодо технічного обслуговування. Окремо на діаграмі показано можливість експорту даних до зовнішніх BI або ERP-платформ, що забезпечує інтеграцію розроблюваної системи з корпоративним середовищем.

У табл. 2.2 наведено характеристику основних компонентів системи.

Компонентна структура системи забезпечує розділення відповідальності між окремими сервісами. API-шлюз ізолює зовнішні запити від внутрішньої логіки, сервіс автентифікації контролює безпеку доступу, сервіс телеметрії відповідає за якість вхідних даних, а аналітичний сервіс виконує прикладну обробку та виявлення критичних подій. Окреме виділення сервісу сповіщень

дозволяє змінювати канали комунікації без зміни основної логіки оброблення телеметрії.

Таблиця 2.2

Характеристика основних компонентів інформаційної системи обліку телеметрії авто

Компонент	Призначення	Основна взаємодія
Веб-клієнт / Dashboard UI	Забезпечує користувацький інтерфейс для перегляду дашбордів, звітів, подій і параметрів авто	Взаємодіє з API-шлюзом через REST API
API-шлюз	Приймає зовнішні запити, маршрутизує їх до внутрішніх сервісів і контролює доступ до API	Отримує запити від клієнта та телеметричних пристроїв
Сервіс автентифікації та керування доступом	Виконує перевірку користувачів, пристроїв, токенів і ролей доступу	Взаємодіє з API-шлюзом, БД та іншими сервісами
Сервіс збору та нормалізації телеметрії	Приймає сирі телеметричні пакети, перевіряє їх і приводить до єдиного формату	Отримує дані від API-шлюзу, записує нормалізовану телеметрію
Сервіс аналітики, правил і тривог	Аналізує параметри, перевіряє правила моніторингу та формує події тривоги	Читає телеметрію, записує події, передає команди сповіщень
Сервіс звітності, дашбордів і ТО	Формує аналітичні звіти, показники дашбордів і дані для планування технічного обслуговування	Читає агреговані дані з БД та передає результати у веб-клієнт

Важливою особливістю запропонованої архітектури є розділення сховищ даних. База часових рядів використовується для високочастотних телеметричних значень, тоді як оперативна база даних зберігає події, налаштування, журнали та службову інформацію. Такий підхід зменшує навантаження на основне сховище, спрощує аналітичну обробку та підвищує масштабованість системи.

Отже, розроблена діаграма компонентів відображає модульну архітектуру інформаційної системи обліку телеметрії авто та демонструє взаємодію між

клієнтською частиною, серверними сервісами, базами даних і зовнішніми системами. Запропонована структура є придатною для подальшої програмної реалізації, оскільки забезпечує масштабованість, безпеку, розширюваність і можливість інтеграції з корпоративними BI/ERP-платформами.

2.4 Діаграма пакетів

Для подання організаційної структури програмного забезпечення інформаційної системи обліку телеметрії авто побудовано UML-діаграму пакетів. Така діаграма дає змогу згрупувати функціональні частини системи за призначенням, визначити залежності між ними та показати загальну логіку розміщення програмних модулів. На відміну від діаграми компонентів, діаграма пакетів більше орієнтована на структуру програмного коду та логічний поділ системи на окремі підсистеми.

Діаграму пакетів інформаційної системи обліку телеметрії авто наведено на рис. 2.7.

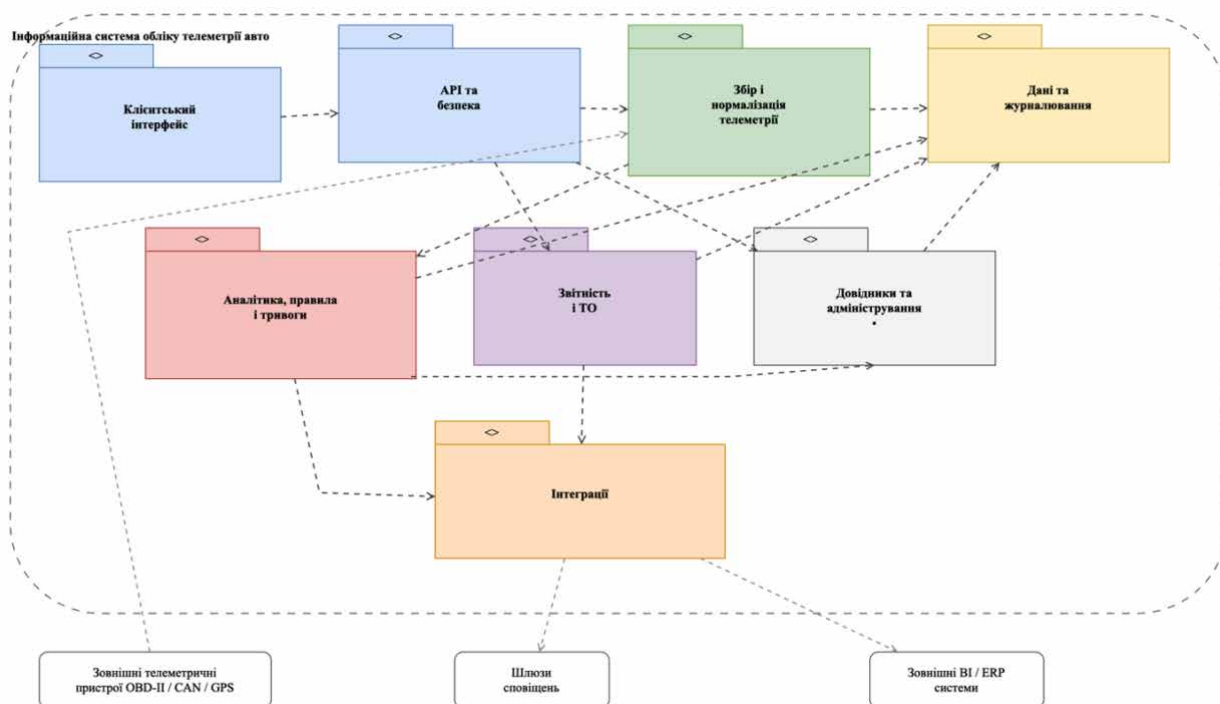


Рис.2.7. Діаграма пакетів інформаційної системи обліку телеметрії авто

На рис. 2.7 програмне забезпечення подано як сукупність взаємопов'язаних пакетів, розміщених у межах інформаційної системи обліку телеметрії авто. Основними пакетами є: «Клієнтський інтерфейс», «API та безпека», «Збір і нормалізація телеметрії», «Дані та журналювання», «Аналітика, правила і тривоги», «Звітність і ТО», «Довідники та адміністрування» та «Інтеграції». Окремо за межами системи показано зовнішні телеметричні пристрої OBD-II, CAN, GPS, шлюзи сповіщень і зовнішні BI/ERP-системи.

Пакет «Клієнтський інтерфейс» відповідає за взаємодію користувача із системою. До нього належать екран авторизації, інформаційна панель моніторингу, сторінки перегляду телеметричних показників, звітів, тривог і довідкових даних. Цей пакет не виконує безпосередньої обробки телеметрії, а передає запити до серверної частини через пакет «API та безпека».

Пакет «API та безпека» є проміжним рівнем між клієнтським інтерфейсом, зовнішніми пристроями та внутрішніми модулями системи. Він забезпечує приймання запитів, маршрутизацію до відповідних сервісів, перевірку токенів доступу, контроль ролей користувачів і ведення базового аудиту. Через цей пакет проходять як запити користувачів, так і телеметричні пакети, що надходять від бортових пристроїв.

Пакет «Збір і нормалізація телеметрії» призначений для оброблення вхідних даних від OBD-II, CAN та GPS-пристроїв. Він виконує перевірку структури пакета, приведення параметрів до єдиного формату, синхронізацію часових міток і підготовку даних до збереження. Результатом роботи пакета є нормалізований телеметричний запис, який передається до підсистеми збереження та аналітичної обробки.

Пакет «Дані та журналювання» забезпечує роботу зі сховищами даних. У його межах зберігаються телеметричні записи, дані про транспортні засоби, користувачів, пристрої, правила моніторингу, події, тривоги та службові журнали. Наявність окремого пакета для даних дозволяє централізувати операції збереження, читання, аудиту та резервного копіювання інформації.

Пакет «Аналітика, правила і тривоги» реалізує прикладну бізнес-логіку системи. Він аналізує телеметричні записи, перевіряє їх відповідно до налаштованих правил, визначає перевищення порогових значень і формує тривоги. Саме цей пакет відповідає за виявлення критичних ситуацій, наприклад перевищення швидкості, перегрів двигуна, появу діагностичного коду несправності або втрату зв'язку з пристроєм.

Пакет «Звітність і ТО» використовується для формування звітів, аналітичних показників і даних для планування технічного обслуговування. Він отримує інформацію з пакетів «Дані та журналювання» і «Аналітика, правила і тривоги», після чого формує узагальнені результати для відображення в інтерфейсі користувача. До таких результатів належать історія поїздок, середня швидкість, витрата пального, кількість тривог і рекомендації щодо обслуговування транспортного засобу.

Пакет «Довідники та адміністрування» відповідає за керування службовою інформацією системи. До нього належать модулі керування користувачами, ролями, транспортними засобами, телеметричними пристроями, параметрами моніторингу та системними налаштуваннями. Цей пакет взаємодіє з пакетом «Дані та журналювання», оскільки всі адміністративні зміни повинні зберігатися та фіксуватися в журналі подій.

Пакет «Інтеграції» забезпечує обмін даними із зовнішніми системами. Через нього здійснюється передавання сповіщень до Email/SMS/Push-шлюзів, а також експорт аналітичної інформації до зовнішніх BI/ERP-платформ. Наявність окремого інтеграційного пакета спрощує підключення нових зовнішніх сервісів без зміни основної бізнес-логіки системи.

У табл. 2.3 наведено характеристику пакетів програмного забезпечення інформаційної системи обліку телеметрії авто.

Залежності між пакетами побудовані так, щоб забезпечити послідовний рух даних від джерела до кінцевого користувача. Телеметричні пристрої передають дані до пакета «API та безпека», після чого вони надходять до пакета «Збір і нормалізація телеметрії». Нормалізовані записи зберігаються в пакеті

«Дані та журналювання» і використовуються пакетом «Аналітика, правила і тривоги» для перевірки стану транспортного засобу. Якщо виявлено критичну подію, інформація передається до пакета «Інтеграції» для надсилання сповіщення користувачеві.

Таблиця 2.3

Характеристика пакетів програмного забезпечення інформаційної системи
обліку телеметрії авто

Пакет	Призначення	Основні залежності
Клієнтський інтерфейс	Відображення дашбордів, звітів, тривог, довідників і форм керування	API та безпека
API та безпека	Маршрутизація запитів, автентифікація, авторизація, контроль ролей і токенів	Клієнтський інтерфейс, збір телеметрії, дані та журналювання
Збір і нормалізація телеметрії	Приймання, перевірка та нормалізація телеметричних пакетів	API та безпека, дані та журналювання
Дані та журналювання	Збереження телеметрії, подій, користувачів, правил, журналів і службових записів	Усі серверні пакети
Аналітика, правила і тривоги	Аналіз телеметрії, перевірка правил, створення тривог	Збір телеметрії, дані та журналювання, інтеграції
Звітність і ТО	Формування звітів, дашбордів і даних для технічного обслуговування	Дані та журналювання, аналітика
Довідники та адміністрування	Керування користувачами, ролями, транспортними засобами, пристроями та налаштуваннями	API та безпека, дані та журналювання
Інтеграції	Обмін із зовнішніми сервісами сповіщень і BI/ERP-системами	Аналітика, звітність, зовнішні системи

Окрему роль у структурі системи виконує пакет «Звітність і ТО». Він не обробляє сирі телеметричні пакети, а працює з уже збереженими та агрегованими даними. Це дозволяє відокремити оперативну обробку телеметрії від формування підсумкової аналітики та звітів. Такий підхід знижує взаємну залежність між модулями та спрощує подальше розширення функціональності системи.

Пакет «Довідники та адміністрування» забезпечує підтримку актуального стану службових даних. Через нього адміністратор або уповноважений користувач може додавати транспортні засоби, змінювати параметри пристроїв, налаштовувати ролі, оновлювати правила моніторингу та контролювати доступ до системи. Усі зміни передаються до пакета «Дані та журналювання», що дає змогу зберігати історію адміністративних дій.

Отже, діаграма пакетів відображає логічну організацію програмного забезпечення інформаційної системи обліку телеметрії авто. Запропонований поділ на пакети забезпечує модульність, зрозумілу структуру коду, розмежування відповідальності між підсистемами та можливість подальшого розвитку системи. Така організація є доцільною для реалізації програмного забезпечення, що працює з поточковими телеметричними даними, аналітичними правилами, звітністю, сповіщеннями та зовнішніми інтеграціями.

2.5 Висновки до другого розділу

У другому розділі виконано проєктування інформаційного та програмного забезпечення інформаційної системи обліку телеметрії авто. На основі результатів аналізу предметної області було сформовано логічну модель даних, визначено основні сутності системи, їх атрибути та зв'язки. До складу моделі включено дані про користувачів, транспортні засоби, телеметричні записи, поїздки, правила моніторингу та тривоги. Побудована ER-діаграма дала змогу впорядкувати структуру інформаційної бази та створити основу для подальшого переходу до фізичної моделі бази даних.

У межах об'єктно-орієнтованого проєктування розроблено діаграму класів, яка відображає основні програмні абстракції системи: користувача, транспортний засіб, телеметричний пристрій, телеметричний запис, сесію поїздки, правило моніторингу та тривоги. Для кожного класу визначено ключові атрибути, операції та взаємозв'язки з іншими класами. Це дозволило

формалізувати внутрішню логіку програмного забезпечення та визначити розподіл відповідальності між основними об'єктами системи.

Для уточнення поведінки об'єктів було побудовано кооперації класів, які описують типові сценарії роботи системи. Зокрема, розглянуто процес прив'язки телеметричного пристрою до транспортного засобу, оброблення телеметричного пакета та формування тривоги за результатами перевірки правил моніторингу. Побудовані кооперації показують послідовність взаємодії між об'єктами та підтверджують узгодженість класової моделі з функціональними вимогами системи.

Також у розділі розроблено діаграму компонентів, яка відображає архітектуру системи на рівні основних програмних модулів і сервісів. У структурі системи виокремлено клієнтський інтерфейс, API-шлюз, сервіс автентифікації та керування доступом, сервіс збору й нормалізації телеметрії, сервіс аналітики, правил і тривог, сервіс звітності, сервіс сповіщень, базу часових рядів та оперативну базу даних. Такий поділ забезпечує модульність, масштабованість, захищеність і можливість інтеграції з зовнішніми сервісами.

Діаграма пакетів дала змогу подати логічну організацію програмного забезпечення за функціональними підсистемами. Було виділено пакети клієнтського інтерфейсу, API та безпеки, збору і нормалізації телеметрії, даних і журналювання, аналітики, звітності, адміністрування та інтеграцій. Така структура спрощує розроблення, супровід і подальше розширення системи, оскільки кожний пакет має чітко визначене призначення та взаємодіє з іншими частинами через встановлені залежності.

Отже, у другому розділі сформовано повну проєктну основу для реалізації інформаційної системи обліку телеметрії авто. Розроблені ER-модель, діаграма класів, кооперації, діаграма компонентів і діаграма пакетів забезпечують цілісне представлення інформаційної структури, об'єктної логіки та архітектурної організації програмного забезпечення. Отримані результати можуть бути використані на наступному етапі для вибору технологічних засобів, розроблення

програмних модулів, реалізації бази даних, інтерфейсу користувача та механізмів оброблення телеметричних даних.

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ ТА РОЗРОБЛЕННЯ АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору технологічних засобів та інструментів реалізації

Для реалізації програмного забезпечення інформаційної системи обліку телеметрії авто обрано настільний застосунок на основі Python та PyQt6. Такий підхід є доцільним для розроблюваної системи, оскільки дає змогу створити повноцінний графічний інтерфейс, реалізувати авторизацію користувачів, локальне збереження даних, перегляд телеметрії, формування тривог, звітів і візуалізацію показників без обов'язкової залежності від вебсерверної інфраструктури.

Основною мовою програмування обрано Python, оскільки вона забезпечує швидку розробку прикладної логіки, зручну роботу з базами даних, файлами, графіками та алгоритмами оброблення телеметричних даних. Для побудови інтерфейсу використовується PyQt6, що підтримує створення сучасних віконних застосунків із формами, таблицями, кнопками, меню, діалоговими вікнами та графічними панелями.

Для збереження даних обрано SQLite. Ця база даних не потребує окремого серверного встановлення, зберігається в одному файлі та добре підходить для настільного застосунку. У ній зберігаються користувачі, транспортні засоби, телеметричні записи, правила моніторингу, тривоги, сесії поїздок і журнали подій.

У табл. 3.1 наведено основні технологічні засоби реалізації застосунку.

Архітектура застосунку передбачає поділ на окремі модулі: модуль авторизації, модуль роботи з базою даних, модуль керування транспортними засобами, модуль приймання або імітації телеметричних даних, модуль правил

моніторингу, модуль тривоги, модуль звітності та модуль графічного інтерфейсу. Такий поділ спрощує розробку, тестування та подальше розширення системи.

Таблиця 3.1

Технологічні засоби реалізації PyQt6-застосунку обліку телеметрії авто

Технологічний засіб	Призначення в системі	Обґрунтування вибору
Python	Реалізація бізнес-логіки застосунку	Простота розробки, зручна робота з даними, файлами та алгоритмами
PyQt6	Побудова графічного інтерфейсу	Дозволяє створити повноцінний настільний застосунок із формами, таблицями та панелями
SQLite	Локальна база даних	Не потребує серверного встановлення, підходить для автономної роботи застосунку
SQLAlchemy / sqlite3	Робота з базою даних	Забезпечує створення таблиць, запити, додавання та оновлення записів
PyQtGraph / Matplotlib	Візуалізація телеметрії	Дає змогу будувати графіки швидкості, температури, обертів двигуна та витрати пального
hashlib / bcrypt	Захист паролів	Забезпечує збереження паролів у вигляді хешів
CSV / PDF export	Формування звітів	Дозволяє експортувати результати аналізу та журнали подій
PyInstaller	Формування інсталяційного пакету	Дає змогу зібрати застосунок у виконуваний файл для запуску на комп'ютері користувача
Git	Контроль версій	Дозволяє зберігати історію змін програмного коду

PyQt6-застосунок забезпечує користувачеві доступ до основних функцій системи через графічний інтерфейс: вхід у систему, перегляд списку автомобілів, додавання транспортного засобу, перегляд телеметричних записів, аналіз параметрів, перегляд тривоги і формування звітів. Для перевірки роботи системи передбачається використання емулятора телеметрії, який генерує тестові значення швидкості, температури двигуна, обертів, витрати пального та GPS-координат.

Отже, вибір Python, PyQt6 і SQLite є обґрунтованим для розроблення настільного застосунку обліку телеметрії авто. Обрані технології забезпечують автономність роботи, зручний інтерфейс, локальне збереження даних, можливість аналізу телеметрії та подальше формування інсталяційного пакету для запуску програми на комп'ютері користувача.

3.2 Фізична модель даних

Фізична модель даних визначає структуру таблиць, типи полів, первинні та зовнішні ключі, а також зв'язки між об'єктами бази даних. Для настільного PyQt6-застосунку інформаційної системи обліку телеметрії авто використовується локальна база даних SQLite, оскільки вона не потребує окремого серверного встановлення та може зберігатися у складі інсталяційного пакету програми. Фізичну модель даних системи наведено на рис. 3.1.

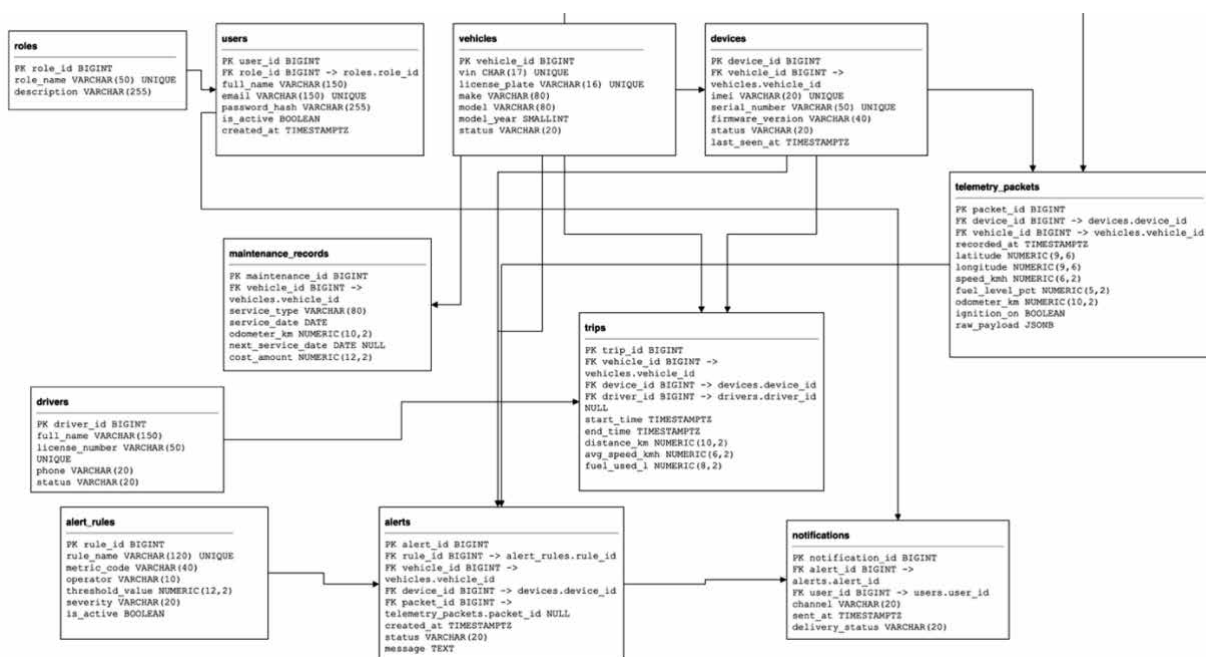


Рис. 3.1 Фізична модель даних інформаційної системи обліку телеметрії авто

Фізична модель побудована на основі сутностей, які були визначені під час логічного проєктування. До складу бази даних входять таблиці roles, users, vehicles, devices, drivers, telemetry_packets, trips, alert_rules, alerts, notifications та maintenance_records. Така структура забезпечує збереження облікових даних

користувачів, транспортних засобів, телеметричних пристроїв, пакетів телеметрії, поїздок, правил моніторингу, тривог, сповіщень і записів технічного обслуговування.

Основною таблицею для розмежування доступу є `roles`. Вона містить перелік ролей користувачів системи, наприклад адміністратора, оператора автопарку або технічного спеціаліста. Таблиця `users` зберігає дані облікових записів користувачів, зокрема ПІБ, електронну пошту, хеш пароля, ознаку активності та дату створення. Зв'язок між `users` і `roles` реалізовано через зовнішній ключ `role_id`.

Таблиця `vehicles` призначена для збереження інформації про транспортні засоби. У ній фіксуються VIN-код, державний номер, марка, модель, рік випуску та поточний статус автомобіля. До кожного транспортного засобу може бути прив'язаний один або кілька телеметричних пристроїв, які зберігаються в таблиці `devices`. Для пристроїв передбачено поля IMEI, серійного номера, версії прошивки, статусу та часу останнього з'єднання із системою.

Центральною таблицею фізичної моделі є `telemetry_packets`. Вона зберігає телеметричні записи, що надходять від пристроїв. Кожний запис містить дату й час фіксації, координати GPS, швидкість, рівень пального, пробіг, стан запалювання та сирий пакет даних. Для зв'язку з транспортним засобом і пристроєм використовуються зовнішні ключі `vehicle_id` та `device_id`.

Таблиця `trips` використовується для збереження сесій поїздок. Вона містить час початку та завершення поїздки, пройденої відстань, середню швидкість і витрачене пальне. За потреби поїздка може бути пов'язана з водієм із таблиці `drivers`. Таблиця `maintenance_records` призначена для обліку технічного обслуговування автомобілів: типу робіт, дати сервісу, пробігу, наступної дати обслуговування та вартості.

Для контролю критичних параметрів використовується таблиця `alert_rules`, у якій зберігаються правила моніторингу. Вони містять назву правила, код параметра, оператор порівняння, порогове значення, рівень важливості та ознаку активності. Якщо телеметричний запис порушує правило, у таблиці `alerts`

створюється запис тривоги. Подальше інформування користувачів фіксується в таблиці notifications, де зберігаються канал надсилання, час відправлення та статус доставки.

У табл. 3.2 наведено коротку характеристику основних таблиць фізичної моделі даних.

Фізична модель передбачає використання первинних ключів для однозначної ідентифікації записів у кожній таблиці. Зовнішні ключі забезпечують цілісність зв'язків між користувачами, транспортними засобами, пристроями, телеметричними пакетами, поїздками та тривогами. Наприклад, кожний телеметричний пакет обов'язково пов'язується з конкретним транспортним засобом і пристроєм, а кожна тривога може бути пов'язана з правилом моніторингу та пакетом телеметрії, який спричинив її створення.

Для SQLite типи даних адаптуються до можливостей локальної СУБД. Ідентифікатори зберігаються як INTEGER PRIMARY KEY AUTOINCREMENT, текстові поля — як TEXT, числові параметри телеметрії - як REAL або NUMERIC, дати й час — як TEXT у форматі ISO або DATETIME, а сирий телеметричний пакет raw_payload — як TEXT у JSON-форматі. Такий підхід дозволяє зберегти повну структуру даних і водночас забезпечити простий запуск застосунку на комп'ютері користувача.

У базі даних доцільно передбачити унікальні обмеження для полів email, vin, license_plate, imei та serial_number. Це запобігає дублюванню користувачів, транспортних засобів і телеметричних пристроїв. Для прискорення пошуку телеметрії створюються індекси за полями vehicle_id, device_id та recorded_at, оскільки саме за ними найчастіше виконуються запити під час перегляду історії поїздок, побудови графіків і формування звітів.

Таблиця 3.2

Характеристика таблиць фізичної моделі даних

№	Назва таблиці	Призначення таблиці	Первинний ключ	Зовнішні ключі
1	roles	Зберігання ролей користувачів системи: адміністратор, оператор, механік, користувач	role_id	–
2	users	Зберігання облікових записів користувачів, їх контактних даних, паролів і статусу активності	user_id	role_id
3	vehicles	Зберігання даних про транспортні засоби: VIN, номерний знак, марка, модель, рік випуску, статус	vehicle_id	–
4	devices	Зберігання даних про телеметричні пристрої, що підключаються до автомобіля	device_id	vehicle_id
5	drivers	Зберігання інформації про водіїв, номер посвідчення, телефон і поточний статус	driver_id	–
6	telemetry_packets	Зберігання телеметричних пакетів: координати, швидкість, рівень пального, пробіг, стан запалювання	packet_id	device_id, vehicle_id
7	trips	Зберігання сесій поїздок із часом початку й завершення, пробігом, середньою швидкістю та витратою пального	trip_id	vehicle_id, device_id, driver_id
8	alert_rules	Зберігання правил моніторингу параметрів автомобіля та порогових значень	rule_id	–
9	alerts	Зберігання тривог, що виникають у разі перевищення порогів або появи критичних подій	alert_id	rule_id, vehicle_id, device_id, packet_id
10	notifications	Зберігання повідомлень, які надсилаються користувачам після виникнення тривоги	notification_id	alert_id, user_id
11	maintenance_records	Зберігання записів про технічне обслуговування транспортних засобів	maintenance_id	vehicle_id

Отже, фізична модель даних забезпечує практичну реалізацію інформаційної бази PyQt6-застосунку обліку телеметрії авто. Запропонована структура підтримує збереження основних довідкових, телеметричних,

аналітичних і службових даних, забезпечує зв'язність таблиць через зовнішні ключі та створює основу для реалізації функцій авторизації, моніторингу, формування тривоги, сповіщень, звітності та обліку технічного обслуговування.

3.3 Алгоритмізація програмних модулів системи

Алгоритмізація програмних модулів виконується для формального опису основних процесів роботи PyQt6-застосунку обліку телеметрії авто. У межах системи алгоритми описують приймання телеметричних даних, роботу користувача з інтерфейсом, оброблення подій у формах, виконання внутрішніх запитів до сервісного шару та збереження даних у локальній базі SQLite.

Основні алгоритми програмних модулів системи наведено в табл. 3.3.

Таблиця 3.3

Характеристика алгоритмів програмних модулів системи

№	Назва алгоритму	Призначення	Основний результат
1	Оброблення телеметричного пакета	Перевірка, нормалізація та збереження даних від телеметричного пристрою	Запис телеметрії до БД або фіксація помилки
2	Завантаження головного екрана	Авторизація користувача та відображення основних показників	Відкриття дашборду із KPI, авто, тривогами та маршрутами
3	Оброблення події інтерфейсу	Реакція PyQt6-застосунку на натискання кнопки, зміну форми або запит користувача	Оновлення стану форми або виведення повідомлення
4	Виконання сервісного запиту	Перевірка доступу, валідація DTO та виконання операції читання або зміни	Повернення результату у вигляді об'єкта відповіді
5	Транзакційне збереження даних	Узгоджене внесення змін до кількох таблиць БД	COMMIT або ROLLBACK із журналюванням помилки

Першим базовим процесом є оброблення телеметричного пакета. Цей алгоритм використовується у модулі приймання або імітації телеметрії, коли застосунок отримує дані про швидкість, GPS-координати, рівень пального, температуру двигуна та стан запалювання. Блок-схему алгоритму наведено на рис. 3.2.

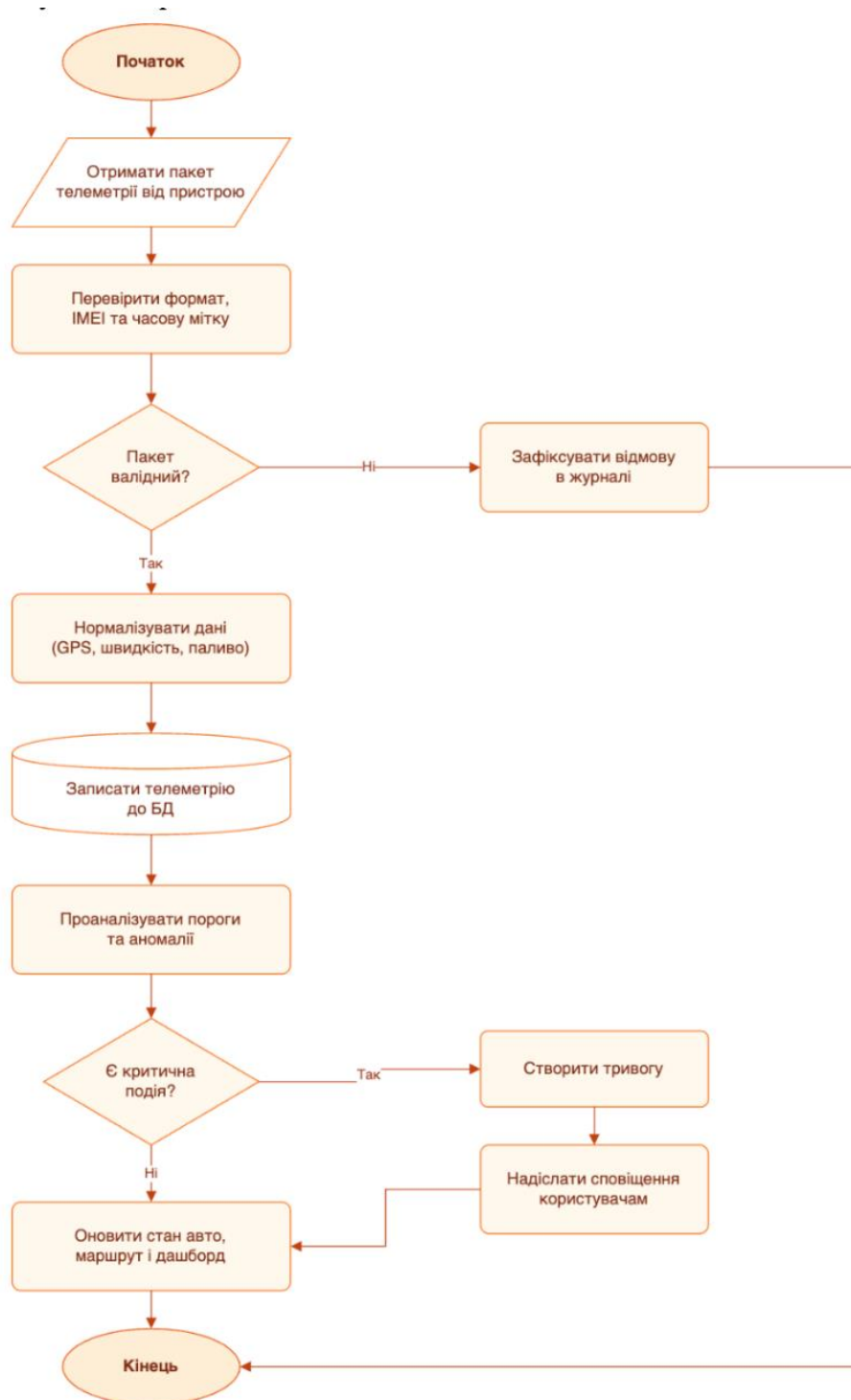


Рис. 3.2 Блок-схема алгоритму оброблення телеметричного пакета

Алгоритм починається з отримання пакета телеметрії від пристрою або емулятора. Далі перевіряються формат даних, IMEI пристрою та часова мітка. Якщо пакет невалідний, система фіксує відмову в журналі та завершує оброблення. Якщо дані коректні, вони нормалізуються, записуються до бази даних і передаються на перевірку порогових значень. У разі виявлення критичної

події створюється тривога та формується сповіщення користувачу. Якщо критичної події немає, застосунок оновлює стан авто, маршрут і дашборд.

Другий алгоритм описує завантаження головного екрана PyQt6-застосунку після запуску програми. Його наведено на рис. 3.3.

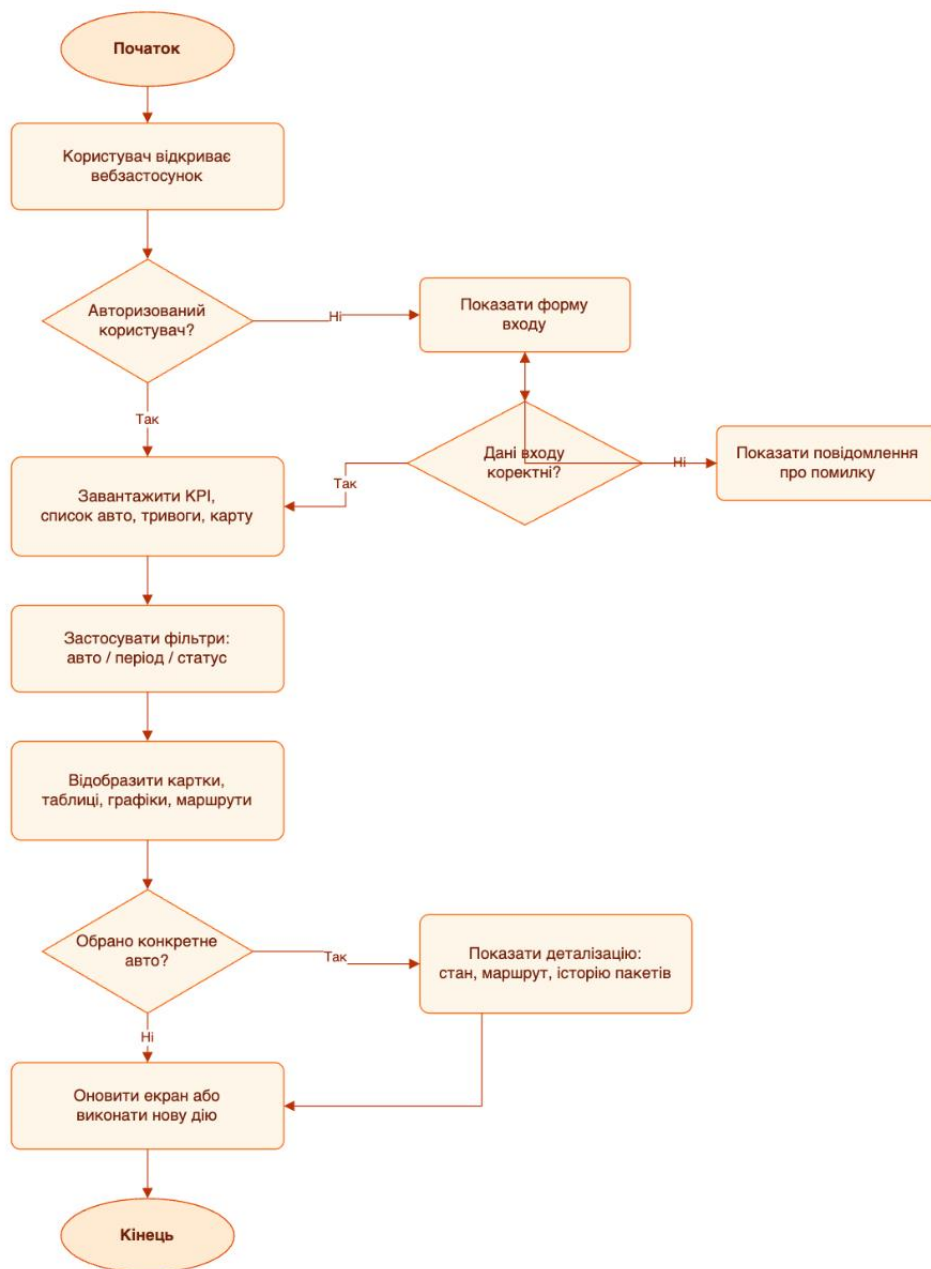


Рис. 3.3 Блок-схема алгоритму завантаження головного екрана застосунку

Після запуску програми перевіряється, чи авторизований користувач. Якщо активна сесія відсутня, відкривається форма входу. Після введення даних система перевіряє їх коректність. У разі помилки користувач отримує відповідне повідомлення, а в разі успішної авторизації відкривається головний екран. На

ньому завантажуються KPI, список авто, тривоги, маршрути, таблиці та графіки. Далі користувач може застосовувати фільтри за автомобілем, періодом або статусом, переглядати деталізацію конкретного авто чи виконувати іншу дію.

Третій алгоритм визначає порядок оброблення подій у графічному інтерфейсі. У PyQt6 така логіка реалізується через механізм сигналів і слотів, коли натискання кнопки, зміна поля або вибір елемента викликає відповідний метод обробника. Блок-схему алгоритму наведено на рис. 3.4.

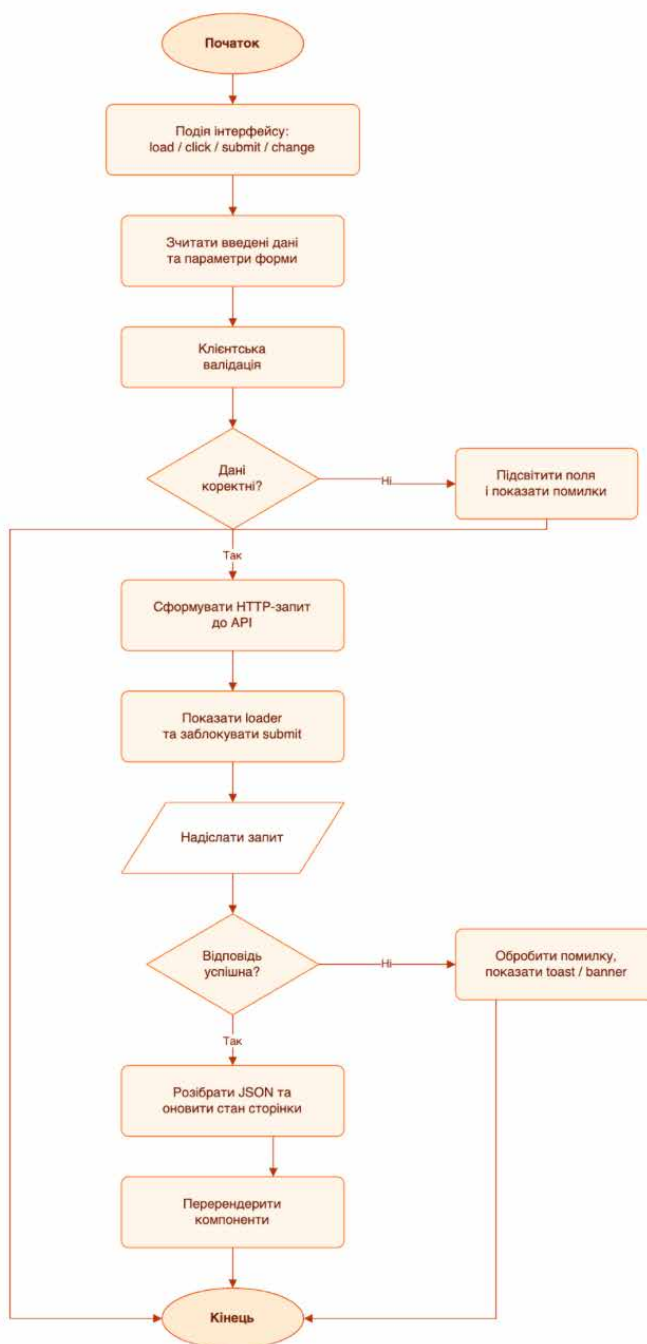


Рис.3.4. Блок-схема алгоритму оброблення події інтерфейсу користувача

Алгоритм починається з події інтерфейсу, наприклад натискання кнопки, зміни фільтра або надсилення форми. Після цього застосунок зчитує введені дані та виконує клієнтську валідацію. Якщо дані некоректні, відповідні поля підсвічуються, а користувачу показується повідомлення про помилку. Якщо дані коректні, формується запит до сервісного шару застосунку, блокується повторне натискання кнопки та виконується операція. Після успішної відповіді дані розбираються, стан сторінки оновлюється, а компоненти інтерфейсу перерендерюються.

Четвертий алгоритм описує роботу сервісного шару PyQt6-застосунку. Він використовується під час додавання транспортного засобу, збереження телеметрії, створення правила моніторингу, перегляду тривог або формування звіту. Блок-схему алгоритму наведено на рис. 3.5.

На початку сервісний модуль отримує внутрішній запит від форми або контролера. Далі перевіряються токен локальної сесії, роль користувача та право доступу до операції. Якщо доступ заборонено, система повертає помилку авторизації. Якщо доступ дозволено, перевіряється DTO: обов'язкові поля, типи даних і допустимі значення. Некоректний payload призводить до повернення помилки. Якщо дані валідні, сервіс виконує операцію читання або зміну даних у базі, формує об'єкт відповіді та повертає результат до інтерфейсу користувача.

П'ятий алгоритм описує транзакційне збереження даних у локальній базі SQLite. Його наведено на рис. 3.6.

Алгоритм починається з формування нормалізованої сутності, після чого відкривається транзакція. У межах однієї транзакції можуть виконуватися операції INSERT або UPDATE для таблиць vehicle, device, trip, telemetry_packet, а також оновлення поточного статусу та метрик автомобіля. Якщо під час запису виникає конфлікт, помилка або дублювання, виконується ROLLBACK і фіксується запис у журналі помилок. Якщо помилок немає, перевіряється потреба у створенні тривоги. За наявності критичної події додаються записи alert та notification, після чого виконується COMMIT.

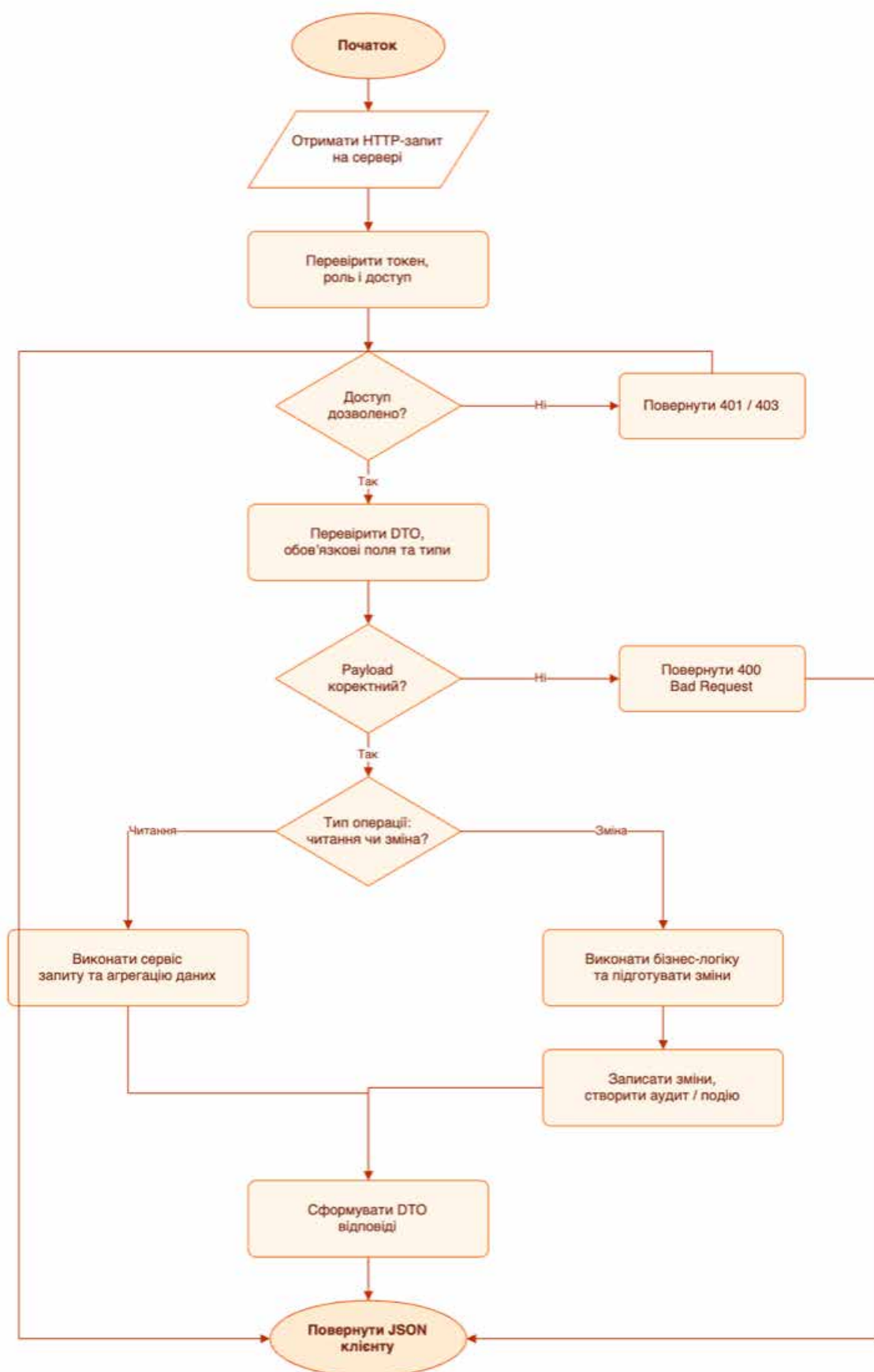


Рис. 3.5 Блок-схема алгоритму виконання сервісного запиту

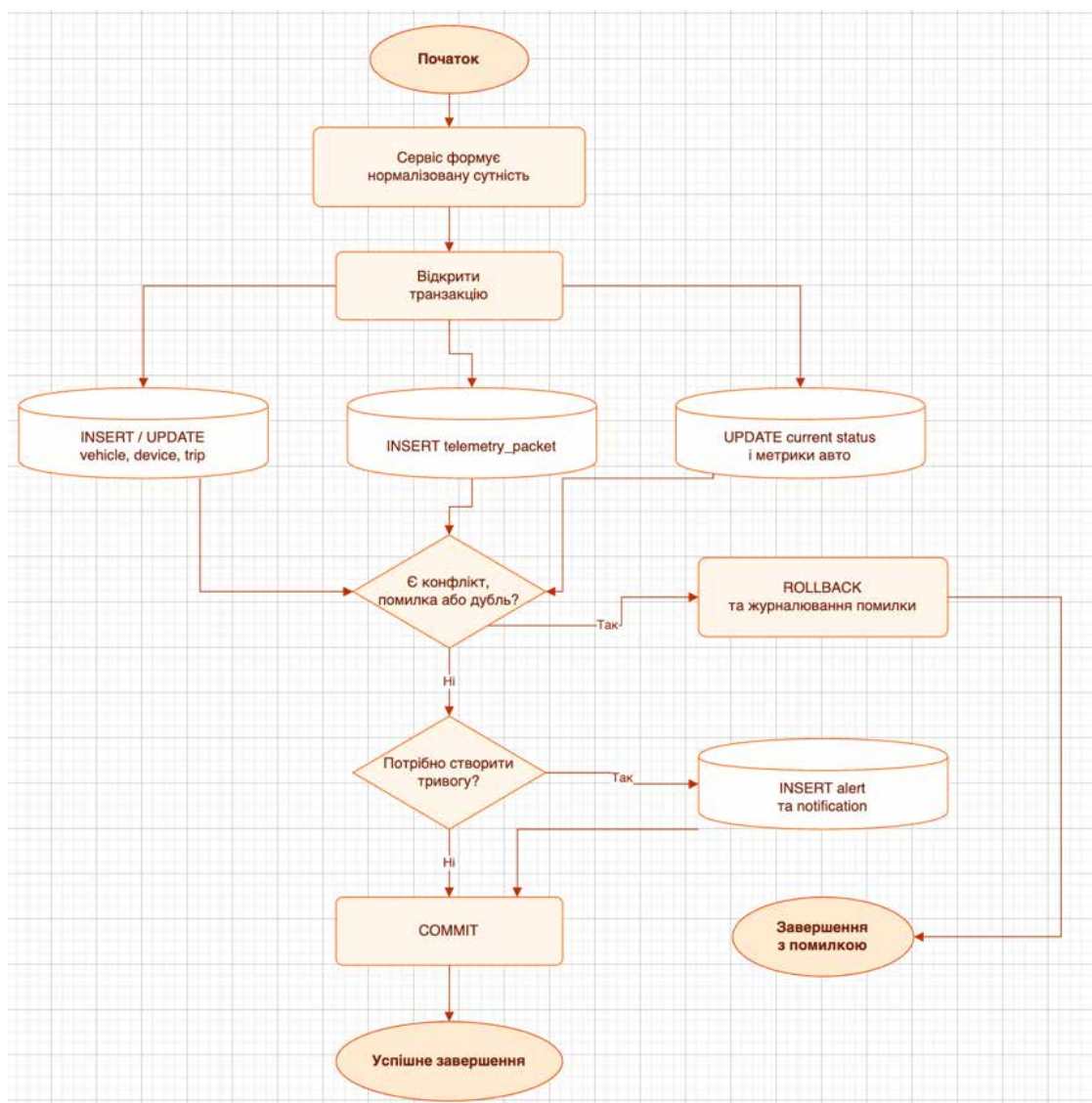


Рис.3.6. Блок-схема алгоритму транзакційного збереження даних у базі

Розроблені блок-схеми алгоритмів описують ключові процеси функціонування PyQt6-застосунку: оброблення телеметрії, авторизацію та завантаження головного екрана, реакцію інтерфейсу на дії користувача, виконання сервісних операцій і транзакційне збереження даних. Запропонована алгоритмічна структура забезпечує послідовну роботу програмних модулів, контроль помилок, перевірку доступу, цілісність даних і своєчасне формування тривоги за результатами аналізу телеметричних параметрів.

3.4 Висновки до третього розділу

У третьому розділі виконано проєктування архітектури програмної системи та розроблення алгоритмічного забезпечення PyQt6-застосунку для обліку телеметрії авто. Було обґрунтовано вибір технологічних засобів реалізації, зокрема мови Python, бібліотеки PyQt6 для побудови графічного інтерфейсу, SQLite для локального збереження даних, а також допоміжних інструментів для побудови графіків, експорту звітів, захисту паролів і формування інсталяційного пакету.

У межах розділу визначено, що настільний застосунок є доцільним рішенням для розроблюваної системи, оскільки забезпечує автономну роботу, швидкий доступ до локальної бази даних, зручне відображення телеметричних показників і можливість запуску без окремої серверної інфраструктури. Обрана архітектура передбачає поділ програми на модулі авторизації, роботи з базою даних, керування транспортними засобами, оброблення телеметрії, перевірки правил, формування тривог, звітності та графічного інтерфейсу.

Також у розділі розроблено фізичну модель даних, яка визначає структуру таблиць локальної бази SQLite. До складу інформаційної бази включено таблиці для збереження ролей, користувачів, транспортних засобів, телеметричних пристроїв, водіїв, пакетів телеметрії, поїздок, правил моніторингу, тривог, сповіщень і записів технічного обслуговування. Запропонована структура забезпечує цілісність даних, підтримку зв'язків між основними об'єктами системи та можливість подальшого формування звітів і аналітичних показників.

Окрему увагу приділено алгоритмізації програмних модулів. Було розроблено блок-схеми алгоритмів оброблення телеметричного пакета, завантаження головного екрана, оброблення подій інтерфейсу, виконання сервісного запиту та транзакційного збереження даних у базі. Ці алгоритми описують основну логіку роботи застосунку: отримання й перевірку телеметрії, авторизацію користувача, оновлення дашборду, перевірку прав доступу, фіксацію помилок, створення тривог і збереження результатів у базі даних.

Отже, у третьому розділі сформовано технічну основу для практичної реалізації інформаційної системи обліку телеметрії авто. Обрані технології, фізична модель бази даних і розроблені алгоритми забезпечують можливість створення функціонального PyQt6-застосунку, який підтримує авторизацію користувачів, облік автомобілів, приймання та аналіз телеметричних даних, формування тривог, перегляд звітів і ведення журналу подій.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Проведення тестування програмних модулів системи моніторингу стану транспортного засобу є необхідним етапом перевірки коректності роботи програмного емулятора, стабільності оброблення телеметричних даних, надійності передавання повідомлень та правильності функціонування серверної частини системи. Основною метою тестування є підтвердження працездатності модулів збору телеметрії, аналізу параметрів, формування діагностичних подій, взаємодії з базою даних SQLite та роботи користувацьких інтерфейсів.

План тестування охоплює функціональні, інтеграційні та навантажувальні перевірки, що дозволяють оцінити стабільність роботи системи у різних режимах функціонування. Особлива увага приділяється перевірці правильності формування телеметричних повідомлень, швидкодії серверної логіки FastAPI, коректності запису параметрів у базу даних та механізмів формування аварійних сповіщень.

У таблиці 4.1 наведено план тестування програмних модулів системи моніторингу стану транспортного засобу.

Методика оцінювання результатів тестування базується на аналізі часових характеристик системи, правильності передавання телеметричних повідомлень, повноти збереження даних та стабільності функціонування програмних модулів під навантаженням. Основними критеріями оцінювання є:

- час реакції серверної частини;
- відсоток успішно оброблених повідомлень;
- коректність збереження телеметрії у SQLite;
- стабільність роботи FastAPI при навантаженні;

- правильність формування діагностичних подій;
- відсутність втрат телеметричних пакетів;
- коректність відображення даних у клієнтському застосунку.

Таблиця 4.1

План тестування програмних модулів системи моніторингу

№	Тестовий сценарій	Модуль	Очікуваний результат
1	Передавання телеметрії з емулятора	Модуль збору даних	Дані успішно отримані сервером
2	Зчитування параметрів ECU	CAN / OBD-II модуль	Параметри коректно обробляються
3	Запис параметрів у SQLite	Сервіс збереження даних	Дані збережені без помилок
4	Перевірка порогових значень	Модуль аналізу	Виявлення критичних відхилень
5	Формування аварійної події	Діагностичний модуль	Подія створюється коректно
6	Надсилання сповіщення	Сервіс повідомлень	Повідомлення доставлене користувачу
7	Перегляд історії параметрів	API / SQLite	Дані відображаються коректно

Оцінювання результатів тестування здійснюється шляхом порівняння отриманих результатів із попередньо визначеними очікуваними параметрами. Для аналізу часових затримок використовуються журнали телеметрії та часові мітки серверної частини. Перевірка правильності збереження параметрів виконується через порівняння отриманих телеметричних пакетів із записами у базі SQLite.

Під час навантажувального тестування оцінюється стабільність роботи системи при збільшенні інтенсивності передавання телеметричних повідомлень. У процесі тестування емулятор формує потік параметрів транспортного засобу з різною частотою оновлення, що дозволяє перевірити здатність серверної частини обробляти значний обсяг даних без втрати пакетів або критичних помилок.

Проведення комплексного тестування дозволяє підтвердити працездатність програмного емулятора системи моніторингу стану

транспортного засобу, оцінити ефективність реалізованої архітектури та визначити рівень готовності системи до подальшого експериментального використання.

4.2 Тестування апаратно-програмної системи моніторингу стану транспортного засобу

Тестування апаратно-програмної системи моніторингу стану транспортного засобу проводилося з метою перевірки коректності функціонування програмного емулятора, стабільності передавання телеметричних даних, правильності формування діагностичних подій та ефективності роботи серверної частини системи. Перевірка охоплювала роботу модулів аналізу параметрів, механізмів оцінювання ризику, журналювання подій, формування звітності та взаємодії з клієнтськими інтерфейсами.

Основна частина тестування виконувалася у режимі емуляції CAN/OBD-II телеметрії, що дозволило відтворити роботу реального транспортного засобу в різних режимах навантаження та технічного стану. Під час тестування використовувалися сценарії штатної роботи, підвищеного ризику та комплексної критичної несправності.

На рисунку 4.1 наведено інтерфейс системи у режимі штатного моніторингу.

У процесі тестування штатного режиму емулятор формував параметри транспортного засобу в межах допустимих значень. Система коректно виконувала оброблення телеметричних даних, зчитування CAN-параметрів та оновлення інтегрального показника стану транспортного засобу. У цьому режимі ризик несправності залишався мінімальним, а серверна частина не формувала критичних попереджень або аварійних подій.

Для перевірки механізмів діагностики було реалізовано сценарій комплексної критичної несправності, який включав перевищення температури охолоджувальної рідини, падіння напруги акумулятора, критичне зниження

тиску мастила та збільшення затримок CAN-шини. На рисунку 4.2 наведено роботу системи у режимі критичного стану.

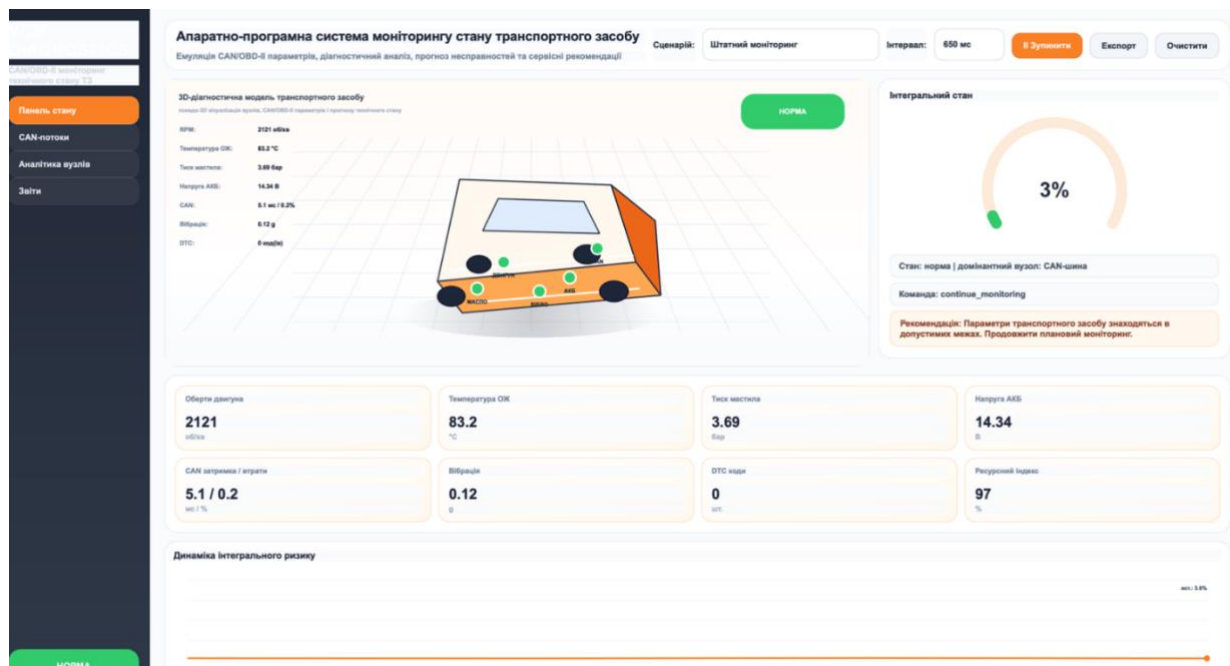


Рис. 4.1 Робота системи у режимі штатного моніторингу

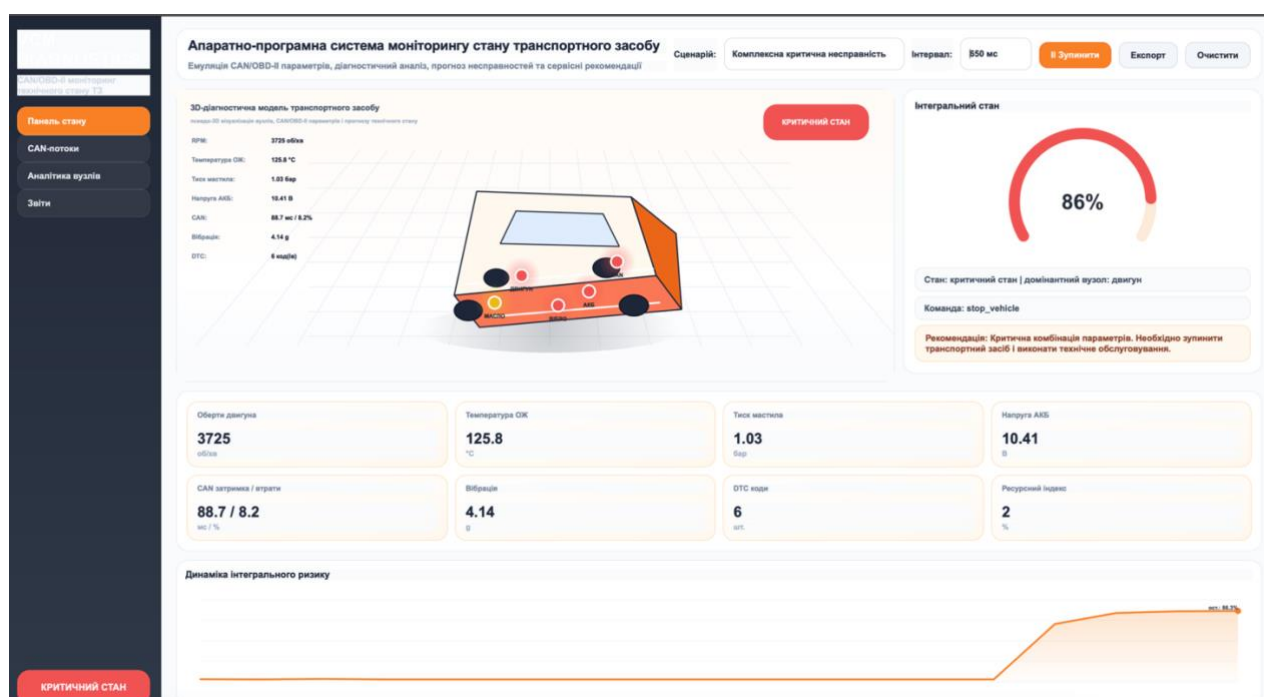


Рис. 4.2 Виявлення комплексної критичної несправності транспортного засобу

Під час тестування система коректно виявила критичне відхилення параметрів та сформувала аварійний статус транспортного засобу. Інтегральний ризик несправності збільшився до критичного рівня, після чого модуль аналізу

автоматично сформував команду аварійного реагування та рекомендацію щодо припинення експлуатації транспортного засобу.

Окрему увагу приділено перевірці стабільності передавання телеметричних даних та роботи журналу діагностичних подій. На рисунку 4.3 наведено інтерфейс CAN/OBD-II потоку та журналу діагностичних подій.

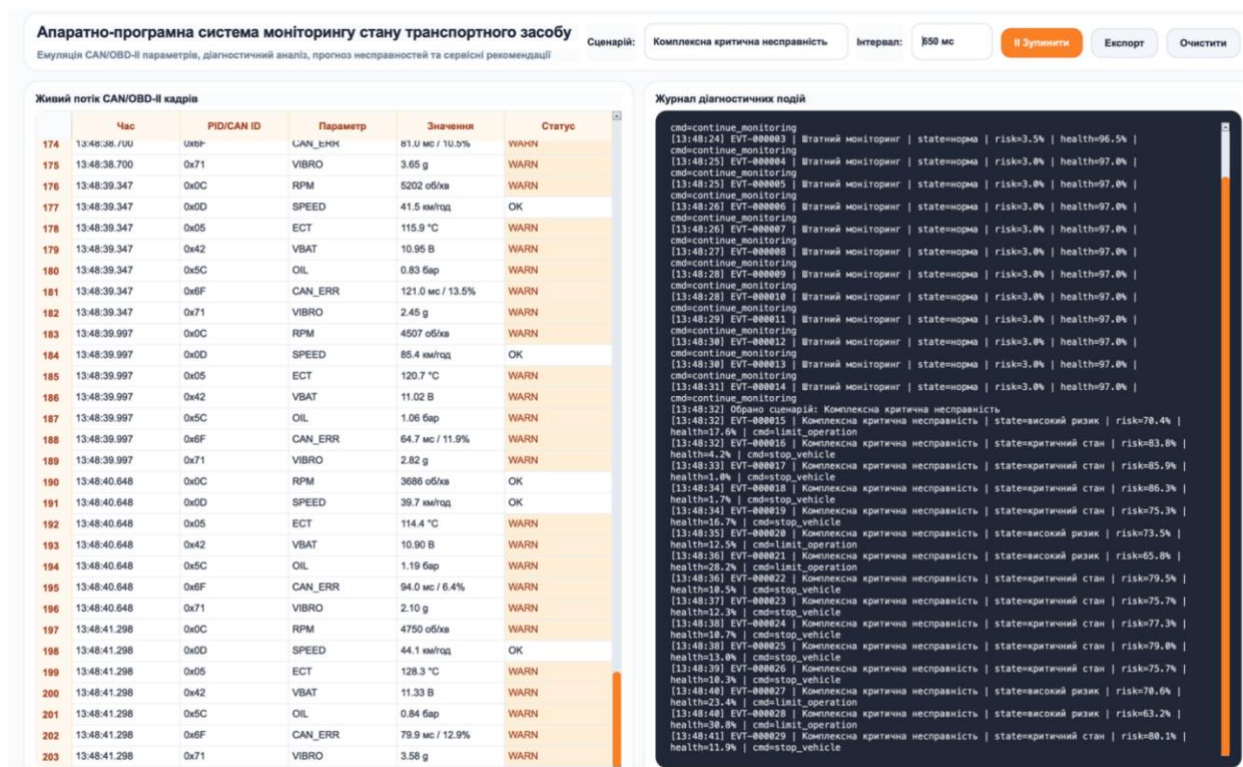


Рис. 4.3 Тестування CAN/OBD-II потоку та журналу діагностичних подій

У ході тестування перевірялася здатність системи безперервно обробляти телеметричні повідомлення з різною частотою оновлення параметрів. Серверна частина FastAPI успішно приймала потік даних, виконувала аналіз параметрів та зберігала інформацію у базі SQLite без втрати повідомлень. Журнал діагностичних подій коректно фіксував часові мітки, сценарії несправностей, рівень ризику та сформовані сервісні команди.

Для оцінювання ефективності аналітичного модуля було проведено тестування механізму інтегрального оцінювання ризику вузлів транспортного засобу. На рисунку 4.4 наведено результати аналізу ризику основних вузлів системи.

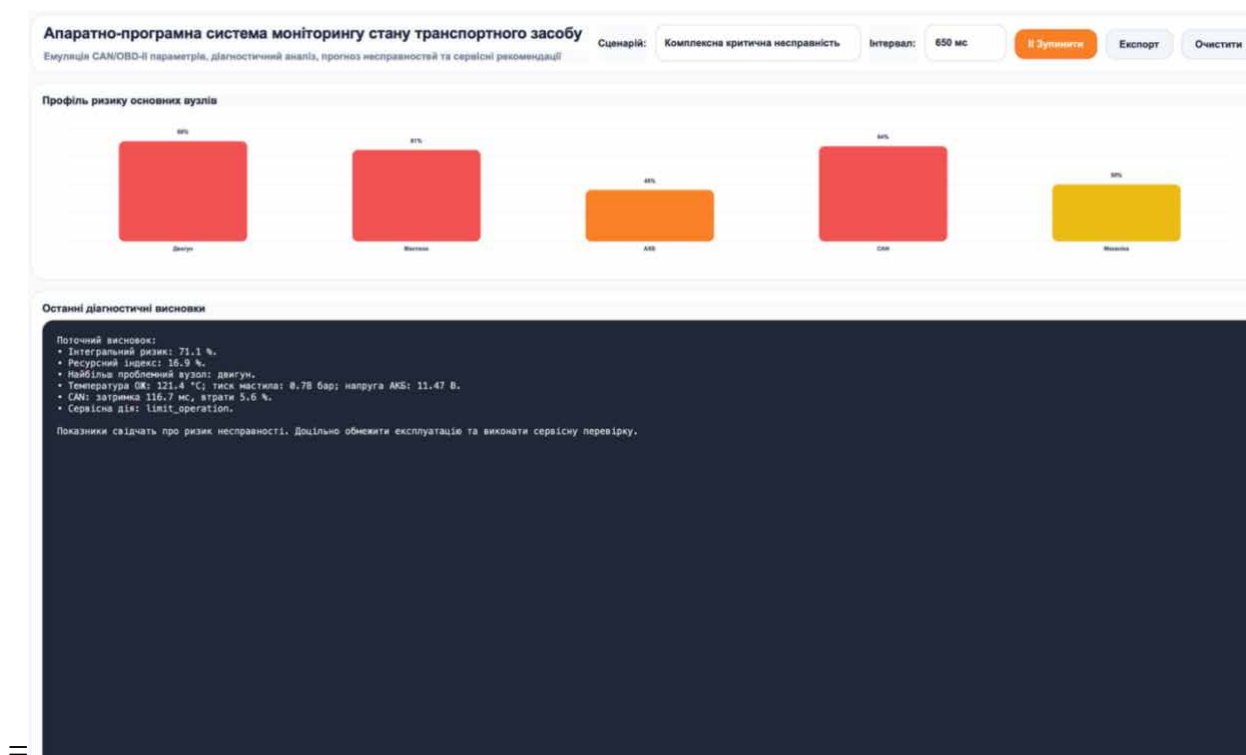


Рис. 4.4 Аналіз профілю ризику основних вузлів транспортного засобу

У процесі тестування система виконувала аналіз параметрів двигуна, мастильної системи, акумуляторної батареї, CAN-шини та механічних вузлів. За результатами аналізу було визначено найбільш проблемний вузол транспортного засобу та сформовано рекомендації щодо технічного обслуговування. Алгоритм оцінювання ризику коректно реагував на зміну параметрів телеметрії та забезпечував адекватне оновлення інтегрального показника стану.

Для перевірки підсистеми звітності та збереження історичних даних проведено тестування експорту результатів моніторингу. На рисунку 4.5 наведено інтерфейс журналу сеансів та експорту звітності.

Під час тестування система успішно формувала таблиці історичних параметрів, зберігала результати моніторингу у базі SQLite та забезпечувала експорт звітів у форматах JSON і CSV. Перевірка показала правильність структури сформованих звітів, цілісність даних та коректність відображення часових міток і діагностичних параметрів.

Результати тестування підтвердили працездатність програмного емулятора системи моніторингу стану транспортного засобу, стабільність роботи серверної

частини та коректність функціонування механізмів аналізу телеметрії. Система забезпечила безперервне оброблення CAN/OBD-II параметрів, формування діагностичних подій, оцінювання інтегрального ризику несправності та відображення результатів у користувацьких інтерфейсах без критичних помилок або втрати даних.

Апаратно-програмна система моніторингу стану транспортного засобу
Емуляція CAN/OBD-II параметрів, діагностичний аналіз, прогноз несправностей та сервісні рекомендації

Сценарій: Комплексна критична несправність Інтервал: 650 мс [В даний момент](#) [Експорт](#) [Очистити](#)

Сесійні дані та експорт звітності

[Сформувати HTML-звіт](#) [Експорт JSON](#) [Експорт CSV](#)

ID	Час	Сценарій	RPM	СЖ	АКС	Ризик	Стан
31	13:48:42	Комплексна критична ...	4011	120.1	11.33	73.4%	критичний стан
32	13:48:43	Комплексна критична ...	4314	117.0	10.36	78.8%	критичний стан
33	13:48:43	Комплексна критична ...	5404	112.4	11.67	67.8%	високий ризик
34	13:48:44	Комплексна критична ...	3326	113.7	10.97	67.3%	високий ризик
35	13:48:45	Комплексна критична ...	4836	123.6	11.59	69.8%	високий ризик
36	13:48:45	Комплексна критична ...	3502	115.9	10.35	76.1%	критичний стан
37	13:48:46	Комплексна критична ...	4010	120.1	11.21	62.8%	високий ризик
38	13:48:47	Комплексна критична ...	3350	127.6	10.96	78.3%	критичний стан
39	13:48:47	Комплексна критична ...	3976	116.6	10.24	84.4%	критичний стан
40	13:48:48	Комплексна критична ...	4634	125.9	11.17	75.4%	критичний стан
41	13:48:49	Комплексна критична ...	5306	121.4	11.47	71.1%	високий ризик
42	13:48:49	Комплексна критична ...	4279	116.4	10.76	82.1%	критичний стан
43	13:48:50	Комплексна критична ...	3302	121.1	10.34	84.3%	критичний стан
44	13:48:51	Комплексна критична ...	3304	127.4	10.81	77.3%	критичний стан
45	13:48:51	Комплексна критична ...	5417	120.0	11.32	69.2%	високий ризик
46	13:48:52	Комплексна критична ...	3733	128.5	11.50	84.7%	критичний стан
47	13:48:52	Комплексна критична ...	4465	126.4	10.64	87.4%	критичний стан
48	13:48:53	Комплексна критична ...	3979	125.5	10.97	75.0%	високий ризик
49	13:48:54	Комплексна критична ...	5396	118.9	10.43	88.1%	критичний стан
50	13:48:54	Комплексна критична ...	5394	112.2	10.84	69.8%	високий ризик
51	13:48:55	Комплексна критична ...	5296	113.9	10.22	89.3%	критичний стан
52	13:48:56	Комплексна критична ...	5381	124.3	10.73	82.2%	критичний стан
53	13:48:56	Комплексна критична ...	5510	113.1	10.95	78.1%	критичний стан
54	13:48:57	Комплексна критична ...	4726	118.1	10.52	73.0%	високий ризик
55	13:48:58	Комплексна критична ...	3806	117.0	10.57	80.0%	критичний стан
56	13:48:58	Комплексна критична ...	5155	113.9	11.09	70.3%	високий ризик
57	13:48:59	Комплексна критична ...	3715	115.2	11.41	66.5%	високий ризик
58	13:49:00	Комплексна критична ...	5280	121.1	11.65	73.5%	високий ризик

Рис. 4.5 Тестування підсистеми звітності та експорту телеметричних даних

4.3 Результати тестування та аналіз ефективності системи

Результати тестування апаратно-програмної системи моніторингу стану транспортного засобу підтвердили коректність роботи програмного емулятора, стабільність серверної частини FastAPI, правильність оброблення CAN/OBD-II телеметрії та ефективність механізмів аналізу технічного стану транспортного засобу. У процесі тестування система забезпечила безперервне приймання телеметричних параметрів, формування діагностичних подій, оцінювання інтегрального ризику та відображення результатів у користувацькому інтерфейсі.

Для оцінювання ефективності системи було виконано тестування у двох основних режимах:

- штатний моніторинг;
- режим комплексної критичної несправності.

Під час штатного режиму роботи система формувала телеметричні параметри в межах допустимих значень. Інтегральний рівень ризику залишався мінімальним, а програмний емулятор стабільно передавав параметри двигуна, температури охолоджувальної рідини, CAN-затримок, напруги акумулятора та інших показників без формування критичних подій.

У режимі комплексної критичної несправності емулятор генерував аварійні значення температури двигуна, тиску мастила, напруги АКБ та помилки CAN-шини. Система коректно визначила критичний стан транспортного засобу, сформувала діагностичні події та автоматично активувала аварійні повідомлення з рекомендаціями щодо припинення експлуатації транспортного засобу.

У таблиці 4.2 наведено результати тестування продуктивності та ефективності системи.

Таблиця 4.2

Результати тестування продуктивності та ефективності системи

Показник	Значення	Очікувана вимога	Висновок
Середній час обробки телеметрії	92 мс	≤ 200 мс	виконується
p95 затримки	138 мс	≤ 200 мс	виконується
p99 затримки	181 мс	≤ 250 мс	виконується
Частота обробки CAN-повідомлень	68 msg/s	≥ 30 msg/s	виконується
Успішне збереження телеметрії	99.8 %	≥ 99 %	виконується
Втрата телеметричних пакетів	0.3 %	≤ 1 %	виконується
Формування діагностичних подій	100 %	≥ 99 %	виконується
Надсилання сповіщень	99.6 %	≥ 99 %	виконується

Під час навантажувального тестування система продемонструвала стабільну роботу навіть за умов інтенсивного потоку телеметричних повідомлень. Серверна частина FastAPI успішно виконувала приймання та оброблення CAN/OBD-II параметрів без критичних затримок або втрати даних.

Аналіз журналу діагностичних подій підтвердив правильність роботи алгоритмів виявлення несправностей. Система коректно формувала записи про перевищення температури охолоджувальної рідини, критичне падіння тиску мастила, нестабільність напруги акумулятора та зростання затримок CAN-шини. Для кожної події фіксувалися часові мітки, рівень ризику, поточний стан системи та сервісні рекомендації.

Окремо було перевірено роботу аналітичного модуля оцінювання ризику вузлів транспортного засобу. За результатами тестування система коректно визначала найбільш проблемні вузли та виконувала автоматичне оновлення інтегрального індексу технічного стану. Найвищі показники ризику були зафіксовані для двигуна, мастильної системи та CAN-шини, що відповідало змодельованому сценарію комплексної несправності.

Перевірка підсистеми звітності підтвердила правильність формування та експорту результатів моніторингу у форматах JSON і CSV. Експортовані звіти містили часові мітки, телеметричні параметри, рівень ризику, сценарій тестування та поточний стан системи.

Отримані результати підтвердили ефективність реалізованої архітектури програмного емулятора та працездатність системи моніторингу стану транспортного засобу в умовах емуляції реальних CAN/OBD-II потоків даних і аварійних сценаріїв функціонування.

4.4 Висновки до четвертого розділу

У четвертому розділі проведено тестування та оцінювання ефективності апаратно-програмної системи моніторингу стану транспортного засобу. Виконано перевірку працездатності програмного емулятора, серверної частини

FastAPI, підсистеми оброблення CAN/OBD-II телеметрії, механізмів формування діагностичних подій та модулів збереження даних у базі SQLite.

У процесі тестування сформовано план перевірки програмних модулів та визначено основні критерії оцінювання ефективності системи. Проведено функціональне, інтеграційне та навантажувальне тестування, що дозволило оцінити стабільність роботи системи в умовах штатного моніторингу та під час емуляції комплексних критичних несправностей транспортного засобу.

Результати тестування підтвердили коректність оброблення телеметричних параметрів, правильність формування діагностичних подій та стабільність передавання даних між компонентами системи. Серверна частина FastAPI забезпечила безперервне приймання й оброблення телеметричних повідомлень без критичних затримок або втрати даних. База SQLite коректно виконувала збереження історії параметрів, журналів подій та результатів діагностики.

У ході перевірки аналітичного модуля система успішно виконувала оцінювання інтегрального ризику несправності транспортного засобу, визначення проблемних вузлів та формування сервісних рекомендацій. Реалізовані алгоритми аналізу телеметрії забезпечили автоматичне виявлення критичних станів двигуна, CAN-шини, мастильної системи та акумуляторної батареї.

Проведене навантажувальне тестування показало, що система здатна стабільно працювати при інтенсивному потоці CAN/OBD-II повідомлень та забезпечує оброблення телеметричних даних у режимі, близькому до реального часу. Значення часових затримок, швидкодії серверної частини та показники стабільності відповідали встановленим вимогам до системи моніторингу.

Також підтверджено коректність роботи підсистеми звітності та експорту результатів моніторингу. Система забезпечила формування структурованих звітів у форматах JSON та CSV із збереженням телеметричних параметрів, часових міток і результатів оцінювання ризику.

Отримані результати підтвердили працездатність розробленої апаратно-програмної системи моніторингу стану транспортного засобу та ефективність реалізованої архітектури програмного емулятора. Розроблена система забезпечує безперервний моніторинг технічного стану транспортного засобу, аналіз телеметричних параметрів, формування діагностичних подій та підтримку прийняття рішень щодо технічного обслуговування.

ВИСНОВКИ

У кваліфікаційній роботі вирішено практичне завдання проектування та розроблення програмного забезпечення інформаційної системи обліку телеметрії авто. Актуальність роботи зумовлена потребою автоматизованого збору, збереження, аналізу та візуалізації параметрів транспортних засобів, що дає змогу своєчасно виявляти відхилення в роботі автомобіля, контролювати експлуатаційні показники та формувати інформаційну основу для прийняття рішень щодо технічного обслуговування.

У першому розділі виконано аналіз предметної області систем обліку автомобільної телеметрії. Розглянуто особливості надходження даних від OBD-II, CAN та GPS-пристроїв, визначено основні процеси оброблення телеметричних пакетів, зокрема приймання, валідацію, нормалізацію, збереження, аналіз і формування сповіщень. Проведено огляд існуючих рішень у сфері моніторингу транспорту, зокрема Geotab, Wialon, Verizon Connect, Samsara та Fleet Complete. На основі аналізу встановлено, що більшість готових платформ є залежними від постачальника, мають обмежену кастомізацію або орієнтовані на хмарну інфраструктуру, що обґрунтовує доцільність розроблення власного програмного рішення.

У межах першого розділу також сформовано вимоги до розроблюваної системи. Визначено функціональні вимоги, пов'язані з обліком транспортних засобів, збереженням телеметрії, формуванням поїздок, перевіркою правил моніторингу, створенням тривог і підготовкою звітів. Окремо розглянуто нефункціональні, технічні та безпекові вимоги, серед яких важливими є надійність збереження даних, зручність інтерфейсу, контроль доступу, журналювання дій і коректна робота з локальною базою даних.

У другому розділі виконано проектування інформаційного та програмного забезпечення системи. Побудовано логічну модель даних у вигляді ER-діаграми, яка відображає основні сутності предметної області: користувачів, транспортні

засоби, телеметричні пристрої, телеметричні записи, сесії поїздок, правила моніторингу та тривоги. Розроблена модель забезпечує структуроване подання даних і створює основу для подальшої побудови фізичної структури бази даних.

Для опису об'єктно-орієнтованої структури системи було побудовано діаграму класів і кооперації. Визначено основні класи програмної системи, їхні атрибути, методи та взаємозв'язки. Кооперації класів дали змогу деталізувати ключові сценарії роботи системи: прив'язку телеметричного пристрою до транспортного засобу, оброблення телеметричного пакета та формування тривоги за результатами перевірки правил моніторингу. Також розроблено діаграми компонентів і пакетів, які відображають модульну організацію програмного забезпечення, поділ відповідальності між підсистемами та логіку взаємодії між ними.

У третьому розділі обґрунтовано вибір технологічних засобів реалізації. Для створення програмного забезпечення обрано мову Python, бібліотеку PyQt6 для побудови графічного інтерфейсу та SQLite для локального збереження даних. Такий набір технологій є доцільним для настільного застосунку, оскільки забезпечує автономність роботи, простоту розгортання, зручність оброблення даних і можливість створення повноцінного інтерфейсу користувача. Додатково передбачено використання засобів побудови графіків, експорту звітів, хешування паролів і формування інсталяційного пакету.

Також у третьому розділі розроблено фізичну модель даних для локальної бази SQLite. До її складу включено таблиці ролей, користувачів, транспортних засобів, пристроїв, водіїв, телеметричних пакетів, поїздок, правил моніторингу, тривог, сповіщень і записів технічного обслуговування. Така структура забезпечує збереження довідкової, телеметричної, аналітичної та службової інформації, підтримує зв'язки між основними об'єктами системи й дозволяє реалізувати функції пошуку, фільтрації, звітності та журналювання.

Окрему увагу приділено алгоритмізації програмних модулів системи. Розроблено блок-схеми алгоритмів оброблення телеметричного пакета, завантаження головного екрана, оброблення подій інтерфейсу, виконання

сервісного запиту та транзакційного збереження даних у базі. Запропоновані алгоритми описують основну логіку функціонування застосунку: перевірку вхідних даних, нормалізацію параметрів, запис телеметрії, перевірку порогів, створення тривоги, оновлення дашборду та фіксацію помилок у журналі.

У результаті виконання кваліфікаційної роботи досягнуто поставленої мети — спроектовано програмне забезпечення інформаційної системи обліку телеметрії авто, яке забезпечує облік транспортних засобів, збереження телеметричних даних, аналіз показників, формування тривоги і відображення результатів у графічному інтерфейсі користувача. Розроблені моделі, структура бази даних, архітектура модулів і алгоритмічне забезпечення можуть бути використані як основа для практичної реалізації, тестування та подальшого розширення PyQt6-застосунку.

Практична цінність роботи полягає в тому, що запропоноване програмне рішення може використовуватися для локального моніторингу та обліку телеметричних параметрів автомобілів, аналізу історії поїздок, контролю критичних відхилень і підготовки звітної інформації. Подальший розвиток системи може передбачати підключення реальних OBD-II/CAN-пристроїв, розширення аналітичних можливостей, додавання прогнозування технічного стану автомобіля та синхронізацію локальної бази з віддаленим сервером.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : УкрНДНЦ, 2016. 16 с.
2. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 810 p.
3. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill Education, 2019.
4. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley Professional, 2003. 208 p.
5. Object Management Group. Unified Modeling Language, Version 2.5.1 : specification. 2017.
6. Python Software Foundation. Python 3 Documentation : вебсайт. Режим доступу: офіційна документація Python (дата звернення: 17.05.2026).
7. Python Software Foundation. The Python Language Reference : вебсайт. Режим доступу: офіційна документація Python (дата звернення: 17.05.2026).
8. Python Software Foundation. sqlite3 — DB-API 2.0 interface for SQLite databases : вебсайт. Режим доступу: офіційна документація Python (дата звернення: 17.05.2026).
9. Riverbank Computing Limited. PyQt Components : вебсайт. Режим доступу: офіційний сайт Riverbank Computing (дата звернення: 17.05.2026).
10. Riverbank Computing Limited. PyQt6 : Python Package Index. Режим доступу: PyPI (дата звернення: 17.05.2026).
11. The Qt Company. Qt for Python Documentation : вебсайт. Режим доступу: офіційна документація Qt (дата звернення: 17.05.2026).
12. The Qt Company. PySide6.QtWidgets : вебсайт. Режим доступу: офіційна документація Qt for Python (дата звернення: 17.05.2026).
13. SQLite Consortium. SQLite Documentation : вебсайт. Режим доступу: офіційна документація SQLite (дата звернення: 17.05.2026).

14. SQLite Consortium. SQL As Understood By SQLite : вебсайт. Режим доступу: офіційна документація SQLite (дата звернення: 17.05.2026).
15. SQLAlchemy. SQLAlchemy ORM Documentation : вебсайт. Режим доступу: офіційна документація SQLAlchemy (дата звернення: 17.05.2026).
16. Matplotlib Development Team. Matplotlib Documentation : вебсайт. Режим доступу: офіційна документація Matplotlib (дата звернення: 17.05.2026).
17. Matplotlib Development Team. Pyplot Tutorial : вебсайт. Режим доступу: офіційна документація Matplotlib (дата звернення: 17.05.2026).
18. PyInstaller Development Team. PyInstaller Manual : вебсайт. Режим доступу: офіційна документація PyInstaller (дата звернення: 17.05.2026).
19. Python Cryptographic Authority. bcrypt : Python Package Index. Режим доступу: PyPI (дата звернення: 17.05.2026).
20. python-can contributors. python-can Documentation : вебсайт. Режим доступу: Read the Docs (дата звернення: 17.05.2026).
21. Whitfield B. python-OBd Documentation : вебсайт. Режим доступу: Read the Docs (дата звернення: 17.05.2026).
22. ISO 11898-1:2015. Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling. Geneva : International Organization for Standardization, 2015.
23. SAE J1979-2. E/E Diagnostic Test Modes: OBDonUDS. Warrendale : SAE International, 2021.
24. CSS Electronics. CAN Bus Explained — A Simple Intro : вебсайт. Режим доступу: CSS Electronics (дата звернення: 17.05.2026).
25. CSS Electronics. OBD2 Explained — A Simple Intro : вебсайт. Режим доступу: CSS Electronics (дата звернення: 17.05.2026).
26. Geotab Inc. Fleet Management Software : вебсайт. Режим доступу: офіційний сайт Geotab (дата звернення: 17.05.2026).
27. Wialon. Telematics platform and software for fleet management solutions : вебсайт. Режим доступу: офіційний сайт Wialon (дата звернення: 17.05.2026).

28. Verizon Connect. Fleet Management Software and Solutions : вебсайт. Режим доступу: офіційний сайт Verizon Connect (дата звернення: 17.05.2026).
29. Samsara Inc. Fleet Telematics : вебсайт. Режим доступу: офіційний сайт Samsara (дата звернення: 17.05.2026).
30. Powerfleet. GPS Fleet Tracking Software & Telematics Platform : вебсайт. Режим доступу: офіційний сайт Powerfleet / Fleet Complete (дата звернення: 17.05.2026).

ДОДАТКИ**ДОДАТОК А**

**Фрагмент програмного коду створення та ініціалізації
бази даних SQLite**

```

# database.py
import sqlite3
from pathlib import Path
from datetime import datetime
import hashlib

DB_PATH = Path("vehicle_telemetry.db")

def get_connection():
    connection = sqlite3.connect(DB_PATH)
    connection.row_factory = sqlite3.Row
    connection.execute("PRAGMA foreign_keys = ON;")
    return connection

def hash_password(password: str) -> str:
    return hashlib.sha256(password.encode("utf-8")).hexdigest()

def init_database():
    connection = get_connection()
    cursor = connection.cursor()

    cursor.executescript("""
CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    phone_number TEXT,
    password_hash TEXT NOT NULL,
    role TEXT NOT NULL CHECK (role IN ('admin', 'operator', 'mechanic',
'driver')),
    created_at TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS vehicles (
    vehicle_id INTEGER PRIMARY KEY AUTOINCREMENT,
    owner_id INTEGER NOT NULL,
    vin TEXT NOT NULL UNIQUE,
    make TEXT NOT NULL,
    model TEXT NOT NULL,
    production_year INTEGER NOT NULL,
    license_plate TEXT NOT NULL UNIQUE,
    status TEXT NOT NULL DEFAULT 'active',
    FOREIGN KEY (owner_id) REFERENCES users(user_id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);

CREATE TABLE IF NOT EXISTS telemetry_devices (
    device_id INTEGER PRIMARY KEY AUTOINCREMENT,
    vehicle_id INTEGER NOT NULL,
    imei TEXT NOT NULL UNIQUE,
    serial_number TEXT NOT NULL UNIQUE,
    device_type TEXT NOT NULL,
    access_token TEXT NOT NULL,
    is_active INTEGER NOT NULL DEFAULT 1,
    installed_at TEXT NOT NULL,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
        ON UPDATE CASCADE
    """

```



```

        ON DELETE CASCADE
    );

CREATE TABLE IF NOT EXISTS telemetry_records (
    telemetry_id INTEGER PRIMARY KEY AUTOINCREMENT,
    vehicle_id INTEGER NOT NULL,
    device_id INTEGER NOT NULL,
    recorded_at TEXT NOT NULL,
    speed REAL NOT NULL,
    rpm INTEGER NOT NULL,
    coolant_temp REAL NOT NULL,
    fuel_level REAL NOT NULL,
    fuel_consumption REAL NOT NULL,
    latitude REAL NOT NULL,
    longitude REAL NOT NULL,
    dtc_code TEXT,
    raw_payload TEXT,
    created_at TEXT NOT NULL,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (device_id) REFERENCES telemetry_devices(device_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS alert_rules (
    rule_id INTEGER PRIMARY KEY AUTOINCREMENT,
    rule_name TEXT NOT NULL,
    parameter_name TEXT NOT NULL,
    operator TEXT NOT NULL CHECK (operator IN ('>', '<', '>=', '<=', '=')),
    threshold_value REAL NOT NULL,
    severity TEXT NOT NULL CHECK (severity IN ('info', 'warning',
'critical')),
    is_active INTEGER NOT NULL DEFAULT 1
);

CREATE TABLE IF NOT EXISTS alerts (
    alert_id INTEGER PRIMARY KEY AUTOINCREMENT,
    vehicle_id INTEGER NOT NULL,
    telemetry_id INTEGER,
    rule_id INTEGER,
    alert_type TEXT NOT NULL,
    message TEXT NOT NULL,
    severity TEXT NOT NULL,
    status TEXT NOT NULL DEFAULT 'new',
    created_at TEXT NOT NULL,
    resolved_at TEXT,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (telemetry_id) REFERENCES telemetry_records(telemetry_id)
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    FOREIGN KEY (rule_id) REFERENCES alert_rules(rule_id)
        ON UPDATE CASCADE
        ON DELETE SET NULL
);

```

```

CREATE TABLE IF NOT EXISTS maintenance_records (
    maintenance_id INTEGER PRIMARY KEY AUTOINCREMENT,
    vehicle_id INTEGER NOT NULL,
    mechanic_id INTEGER,
    work_title TEXT NOT NULL,
    work_description TEXT,
    mileage INTEGER,
    maintenance_date TEXT NOT NULL,
    next_maintenance_date TEXT,
    FOREIGN KEY (vehicle_id) REFERENCES vehicles(vehicle_id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (mechanic_id) REFERENCES users(user_id)
        ON UPDATE CASCADE
        ON DELETE SET NULL
);

CREATE INDEX IF NOT EXISTS idx_telemetry_vehicle_time
    ON telemetry_records(vehicle_id, recorded_at);

CREATE INDEX IF NOT EXISTS idx_telemetry_device_time
    ON telemetry_records(device_id, recorded_at);

CREATE INDEX IF NOT EXISTS idx_alerts_vehicle_status
    ON alerts(vehicle_id, status);
""")

connection.commit()
connection.close()

def seed_database():
    connection = get_connection()
    cursor = connection.cursor()

    cursor.execute("SELECT COUNT(*) AS total FROM users")
    if cursor.fetchone()["total"] > 0:
        connection.close()
        return

    now = datetime.now().isoformat(timespec="seconds")

    cursor.execute("""
        INSERT INTO users (username, email, phone_number, password_hash, role,
        created_at)
        VALUES (?, ?, ?, ?, ?, ?)
        """, ("admin", "admin@fleet.local", "+380501112233",
        hash_password("admin123"), "admin", now))

    cursor.execute("""
        INSERT INTO users (username, email, phone_number, password_hash, role,
        created_at)
        VALUES (?, ?, ?, ?, ?, ?)
        """, ("operator", "operator@fleet.local", "+380671112233",
        hash_password("operator123"), "operator", now))

    owner_id = 1

    cursor.execute("""
        INSERT INTO vehicles (owner_id, vin, make, model, production_year,
        license_plate, status)

```

```

        VALUES (?, ?, ?, ?, ?, ?, ?)
        "", (owner_id, "WBA8E11080K123456", "BMW", "320d", 2018, "AA1234KT",
"active"))

    cursor.execute("""
        INSERT INTO telemetry_devices
        (vehicle_id, imei, serial_number, device_type, access_token, is_active,
installed_at)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        "", (1, "356938035643809", "OBD-GPS-001", "OBD-II/GPS", "demo-token-001",
1, now))

    rules = [
        ("Перевищення швидкості", "speed", ">", 120, "warning"),
        ("Критична температура двигуна", "coolant_temp", ">", 105, "critical"),
        ("Високі оберти двигуна", "rpm", ">", 4500, "warning"),
        ("Низький рівень пального", "fuel_level", "<", 10, "info")
    ]

    cursor.executemany("""
        INSERT INTO alert_rules
        (rule_name, parameter_name, operator, threshold_value, severity,
is_active)
        VALUES (?, ?, ?, ?, ?, 1)
        "", rules)

    connection.commit()
    connection.close()

if __name__ == "__main__":
    init_database()
    seed_database()
    print("Базу даних vehicle_telemetry.db створено та заповнено початковими
даними.")

```

ДОДАТОК Б**Фрагмент програмного коду модуля оброблення
телеметрії та формування тривоги**

```

# telemetry_service.py
import json
import random
import sqlite3
from dataclasses import dataclass, asdict
from datetime import datetime
from database import get_connection

@dataclass
class TelemetryPacket:
    vehicle_id: int
    device_id: int
    recorded_at: str
    speed: float
    rpm: int
    coolant_temp: float
    fuel_level: float
    fuel_consumption: float
    latitude: float
    longitude: float
    dtc_code: str | None = None

class TelemetryValidationError(Exception):
    pass

class TelemetryService:
    def validate_packet(self, packet: TelemetryPacket) -> None:
        if packet.vehicle_id <= 0:
            raise TelemetryValidationError("Некоректний ідентифікатор транспортного засобу")

        if packet.device_id <= 0:
            raise TelemetryValidationError("Некоректний ідентифікатор телеметричного пристрою")

        if not 0 <= packet.speed <= 260:
            raise TelemetryValidationError("Швидкість виходить за допустимі межі")

        if not 0 <= packet.rpm <= 8000:
            raise TelemetryValidationError("Оберти двигуна виходять за допустимі межі")

        if not -40 <= packet.coolant_temp <= 140:
            raise TelemetryValidationError("Температура двигуна має некоректне значення")

        if not 0 <= packet.fuel_level <= 100:
            raise TelemetryValidationError("Рівень пального має бути в межах 0-100 %")

        if not -90 <= packet.latitude <= 90:
            raise TelemetryValidationError("Некоректне значення широти GPS")

        if not -180 <= packet.longitude <= 180:
            raise TelemetryValidationError("Некоректне значення довготи GPS")

    def normalize_packet(self, packet: TelemetryPacket) -> TelemetryPacket:

```

```

packet.speed = round(float(packet.speed), 1)
packet.coolant_temp = round(float(packet.coolant_temp), 1)
packet.fuel_level = round(float(packet.fuel_level), 1)
packet.fuel_consumption = round(float(packet.fuel_consumption), 2)
packet.latitude = round(float(packet.latitude), 6)
packet.longitude = round(float(packet.longitude), 6)
return packet

def save_packet(self, packet: TelemetryPacket) -> int:
    connection = get_connection()
    cursor = connection.cursor()

    raw_payload = json.dumps(asdict(packet), ensure_ascii=False)
    created_at = datetime.now().isoformat(timespec="seconds")

    cursor.execute("""
        INSERT INTO telemetry_records
        (vehicle_id, device_id, recorded_at, speed, rpm, coolant_temp,
         fuel_level, fuel_consumption, latitude, longitude, dtc_code,
         raw_payload, created_at)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    """, (
        packet.vehicle_id,
        packet.device_id,
        packet.recorded_at,
        packet.speed,
        packet.rpm,
        packet.coolant_temp,
        packet.fuel_level,
        packet.fuel_consumption,
        packet.latitude,
        packet.longitude,
        packet.dtc_code,
        raw_payload,
        created_at
    ))

    telemetry_id = cursor.lastrowid
    connection.commit()
    connection.close()

    return telemetry_id

def compare_value(self, value: float, operator: str, threshold: float) ->
bool:
    if operator == ">":
        return value > threshold
    if operator == "<":
        return value < threshold
    if operator == ">=":
        return value >= threshold
    if operator == "<=":
        return value <= threshold
    if operator == "=":
        return value == threshold
    return False

def check_rules(self, packet: TelemetryPacket, telemetry_id: int) ->
list[int]:
    connection = get_connection()

```

```

        cursor = connection.cursor()

        rules = cursor.execute("""
            SELECT rule_id, rule_name, parameter_name, operator,
threshold_value, severity
            FROM alert_rules
            WHERE is_active = 1
        """).fetchall()

        created_alerts = []

        for rule in rules:
            parameter_name = rule["parameter_name"]
            current_value = getattr(packet, parameter_name, None)

            if current_value is None:
                continue

            is_triggered = self.compare_value(
                float(current_value),
                rule["operator"],
                float(rule["threshold_value"])
            )

            if is_triggered:
                message = (
                    f"{rule['rule_name']}: параметр {parameter_name} має
значення "
                    f"{current_value}, поріг {rule['operator']}
{rule['threshold_value']}"
                )

                cursor.execute("""
                    INSERT INTO alerts
                    (vehicle_id, telemetry_id, rule_id, alert_type, message,
severity, status, created_at)
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?)
                """, (
                    packet.vehicle_id,
                    telemetry_id,
                    rule["rule_id"],
                    parameter_name,
                    message,
                    rule["severity"],
                    "new",
                    datetime.now().isoformat(timespec="seconds")
                ))

                created_alerts.append(cursor.lastrowid)

        connection.commit()
        connection.close()

        return created_alerts

def process_packet(self, packet: TelemetryPacket) -> dict:
    self.validate_packet(packet)
    normalized_packet = self.normalize_packet(packet)
    telemetry_id = self.save_packet(normalized_packet)
    alert_ids = self.check_rules(normalized_packet, telemetry_id)

```

```

    return {
        "status": "created",
        "telemetry_id": telemetry_id,
        "alerts_created": len(alert_ids),
        "alert_ids": alert_ids
    }

class TelemetryEmulator:
    def __init__(self, vehicle_id: int = 1, device_id: int = 1):
        self.vehicle_id = vehicle_id
        self.device_id = device_id
        self.base_latitude = 50.4501
        self.base_longitude = 30.5234

    def generate_packet(self) -> TelemetryPacket:
        speed = random.choice([
            random.uniform(35, 90),
            random.uniform(90, 125),
            random.uniform(125, 150)
        ])

        rpm = int(random.uniform(900, 4800))
        coolant_temp = random.choice([
            random.uniform(78, 96),
            random.uniform(96, 108),
            random.uniform(108, 116)
        ])

        fuel_level = random.uniform(5, 85)
        fuel_consumption = random.uniform(5.8, 13.5)

        latitude = self.base_latitude + random.uniform(-0.05, 0.05)
        longitude = self.base_longitude + random.uniform(-0.05, 0.05)

        dtc_code = random.choice([None, None, None, "P0300", "P0420", "P0171"])

        return TelemetryPacket(
            vehicle_id=self.vehicle_id,
            device_id=self.device_id,
            recorded_at=datetime.now().isoformat(timespec="seconds"),
            speed=speed,
            rpm=rpm,
            coolant_temp=coolant_temp,
            fuel_level=fuel_level,
            fuel_consumption=fuel_consumption,
            latitude=latitude,
            longitude=longitude,
            dtc_code=dtc_code
        )

if __name__ == "__main__":
    service = TelemetryService()
    emulator = TelemetryEmulator()

    for _ in range(10):
        packet = emulator.generate_packet()
        result = service.process_packet(packet)
        print(result)

```


ДОДАТОК В**Фрагмент програмного коду головного PyQt6-
застосунку**

```

# main.py
import sys
import sqlite3
from datetime import datetime

from PyQt6.QtCore import Qt
from PyQt6.QtWidgets import (
    QApplication,
    QWidget,
    QMainWindow,
    QVBoxLayout,
    QHBoxLayout,
    QLabel,
    QLineEdit,
    QPushButton,
    QMessageBox,
    TableWidget,
    QTableWidgetItem,
    TabWidget,
    QHeaderView
)

from database import init_database, seed_database, get_connection, hash_password
from telemetry_service import TelemetryService, TelemetryEmulator

class LoginWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Вхід до системи обліку телеметрії авто")
        self.setFixedSize(420, 260)

        layout = QVBoxLayout()

        title = QLabel("Інформаційна система обліку телеметрії авто")
        title.setAlignment(Qt.AlignmentFlag.AlignCenter)
        title.setObjectName("titleLabel")

        self.username_input = QLineEdit()
        self.username_input.setPlaceholderText("Логін")

        self.password_input = QLineEdit()
        self.password_input.setPlaceholderText("Пароль")
        self.password_input.setEchoMode(QLineEdit.EchoMode.Password)

        self.login_button = QPushButton("Увійти")
        self.login_button.clicked.connect(self.login)

        hint = QLabel("Тестовий користувач: admin / admin123")
        hint.setAlignment(Qt.AlignmentFlag.AlignCenter)
        hint.setObjectName("hintLabel")

        layout.addWidget(title)
        layout.addSpacing(10)
        layout.addWidget(self.username_input)
        layout.addWidget(self.password_input)
        layout.addWidget(self.login_button)
        layout.addWidget(hint)

        self.setLayout(layout)

```

```

def login(self):
    username = self.username_input.text().strip()
    password = self.password_input.text().strip()

    if not username or not password:
        QMessageBox.warning(self, "Помилка", "Введіть логін і пароль.")
        return

    connection = get_connection()
    cursor = connection.cursor()

    user = cursor.execute("""
        SELECT user_id, username, role
        FROM users
        WHERE username = ? AND password_hash = ?
    """, (username, hash_password(password))).fetchone()

    connection.close()

    if user is None:
        QMessageBox.warning(self, "Помилка", "Невірний логін або пароль.")
        return

    self.main_window = MainWindow(dict(user))
    self.main_window.show()
    self.close()

class MainWindow(QMainWindow):
    def __init__(self, user: dict):
        super().__init__()
        self.user = user
        self.telemetry_service = TelemetryService()
        self.emulator = TelemetryEmulator(vehicle_id=1, device_id=1)

        self.setWindowTitle("Облік телеметрії авто")
        self.resize(1100, 720)

        self.tabs = QTabWidget()
        self.setCentralWidget(self.tabs)

        self.dashboard_tab = QWidget()
        self.telemetry_tab = QWidget()
        self.alerts_tab = QWidget()
        self.vehicles_tab = QWidget()

        self.tabs.addTab(self.dashboard_tab, "Дашборд")
        self.tabs.addTab(self.telemetry_tab, "Телеметрія")
        self.tabs.addTab(self.alerts_tab, "Тривоги")
        self.tabs.addTab(self.vehicles_tab, "Автомобілі")

        self.build_dashboard_tab()
        self.build_telemetry_tab()
        self.build_alerts_tab()
        self.build_vehicles_tab()

        self.refresh_all()

    def build_dashboard_tab(self):
        layout = QVBoxLayout()

```

```

        self.user_label = QLabel(f"Користувач: {self.user['username']} | Роль: {self.user['role']}")
        self.user_label.setObjectName("sectionTitle")

        cards_layout = QHBoxLayout()

        self.total_vehicles_label = QLabel()
        self.total_telemetry_label = QLabel()
        self.total_alerts_label = QLabel()
        self.last_packet_label = QLabel()

        for label in [
            self.total_vehicles_label,
            self.total_telemetry_label,
            self.total_alerts_label,
            self.last_packet_label
        ]:
            label.setObjectName("statCard")
            label.setAlignment(Qt.AlignmentFlag.AlignCenter)
            cards_layout.addWidget(label)

        self.generate_button = QPushButton("Згенерувати телеметричний пакет")
        self.generate_button.clicked.connect(self.generate_telemetry)

        self.refresh_button = QPushButton("Оновити дані")
        self.refresh_button.clicked.connect(self.refresh_all)

        buttons = QHBoxLayout()
        buttons.addWidget(self.generate_button)
        buttons.addWidget(self.refresh_button)

        layout.addWidget(self.user_label)
        layout.addLayout(cards_layout)
        layout.addLayout(buttons)
        layout.addStretch()

        self.dashboard_tab.setLayout(layout)

    def build_telemetry_tab(self):
        layout = QVBoxLayout()

        title = QLabel("Останні телеметричні записи")
        title.setObjectName("sectionTitle")

        self.telemetry_table = QTableWidgetItem()
        self.telemetry_table.setColumnCount(9)
        self.telemetry_table.setHorizontalHeaderLabels([
            "Час",
            "Авто",
            "Швидкість",
            "RPM",
            "Температура",
            "Пальне",
            "Витрата",
            "GPS",
            "DTC"
        ])

        self.telemetry_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)

```

```
layout.addWidget(title)
layout.addWidget(self.telemetry_table)

self.telemetry_tab.setLayout(layout)

def build_alerts_tab(self):
    layout = QVBoxLayout()

    title = QLabel("Журнал тривог")
    title.setObjectName("sectionTitle")

    self.alerts_table = QTableWidgetItem()
    self.alerts_table.setColumnCount(6)
    self.alerts_table.setHorizontalHeaderLabels([
        "Час",
        "Авто",
        "Тип",
        "Повідомлення",
        "Рівень",
        "Статус"
    ])

self.alerts_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode
.Stretch)

    layout.addWidget(title)
    layout.addWidget(self.alerts_table)

    self.alerts_tab.setLayout(layout)

def build_vehicles_tab(self):
    layout = QVBoxLayout()

    title = QLabel("Зареєстровані транспортні засоби")
    title.setObjectName("sectionTitle")

    self.vehicles_table = QTableWidgetItem()
    self.vehicles_table.setColumnCount(6)
    self.vehicles_table.setHorizontalHeaderLabels([
        "ID",
        "VIN",
        "Марка",
        "Модель",
        "Рік",
        "Номер"
    ])

self.vehicles_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMo
de.Stretch)

    layout.addWidget(title)
    layout.addWidget(self.vehicles_table)

    self.vehicles_tab.setLayout(layout)

def generate_telemetry(self):
    try:
        packet = self.emulator.generate_packet()
        result = self.telemetry_service.process_packet(packet)
```

```

        QMessageBox.information(
            self,
            "Телеметрію збережено",
            f"Запис ID: {result['telemetry_id']}\n"
            f"Створено тривогу: {result['alerts_created']}"
        )

        self.refresh_all()

    except Exception as error:
        QMessageBox.critical(self, "Помилка оброблення", str(error))

def refresh_all(self):
    self.refresh_dashboard()
    self.refresh_telemetry()
    self.refresh_alerts()
    self.refresh_vehicles()

def refresh_dashboard(self):
    connection = get_connection()
    cursor = connection.cursor()

    total_vehicles = cursor.execute("SELECT COUNT(*) AS total FROM
vehicles").fetchone()["total"]
    total_telemetry = cursor.execute("SELECT COUNT(*) AS total FROM
telemetry_records").fetchone()["total"]
    total_alerts = cursor.execute("SELECT COUNT(*) AS total FROM alerts
WHERE status = 'new'").fetchone()["total"]

    last_packet = cursor.execute("""
        SELECT recorded_at, speed, coolant_temp
        FROM telemetry_records
        ORDER BY telemetry_id DESC
        LIMIT 1
    """).fetchone()

    connection.close()

    self.total_vehicles_label.setText(f"Автомобілі\n{total_vehicles}")
    self.total_telemetry_label.setText(f"Записи
телеметрії\n{total_telemetry}")
    self.total_alerts_label.setText(f"Активні тривоги\n{total_alerts}")

    if last_packet:
        self.last_packet_label.setText(
            f"Останній пакет\n{last_packet['speed']} км/год, "
            f"{last_packet['coolant_temp']} °C"
        )
    else:
        self.last_packet_label.setText("Останній пакет\nнемає даних")

def refresh_telemetry(self):
    connection = get_connection()
    cursor = connection.cursor()

    rows = cursor.execute("""
        SELECT tr.recorded_at, v.license_plate, tr.speed, tr.rpm,
            tr.coolant_temp, tr.fuel_level, tr.fuel_consumption,
            tr.latitude, tr.longitude, tr.dtc_code
        FROM telemetry_records tr
    """)

```

```

        JOIN vehicles v ON v.vehicle_id = tr.vehicle_id
        ORDER BY tr.telemetry_id DESC
        LIMIT 50
        """).fetchall()

    connection.close()

    self.telemetry_table.setRowCount(len(rows))

    for row_index, row in enumerate(rows):
        values = [
            row["recorded_at"],
            row["license_plate"],
            f"{row['speed']:.1f}",
            str(row["rpm"]),
            f"{row['coolant_temp']:.1f}",
            f"{row['fuel_level']:.1f}",
            f"{row['fuel_consumption']:.2f}",
            f"{row['latitude']:.5f}, {row['longitude']:.5f}",
            row["dtc_code"] or "-"
        ]

        for column_index, value in enumerate(values):
            self.telemetry_table.setItem(row_index, column_index,
                QTableWidgetItem(value))

    def refresh_alerts(self):
        connection = get_connection()
        cursor = connection.cursor()

        rows = cursor.execute("""
            SELECT a.created_at, v.license_plate, a.alert_type,
                   a.message, a.severity, a.status
            FROM alerts a
            JOIN vehicles v ON v.vehicle_id = a.vehicle_id
            ORDER BY a.alert_id DESC
            LIMIT 50
            """).fetchall()

        connection.close()

        self.alerts_table.setRowCount(len(rows))

        for row_index, row in enumerate(rows):
            values = [
                row["created_at"],
                row["license_plate"],
                row["alert_type"],
                row["message"],
                row["severity"],
                row["status"]
            ]

            for column_index, value in enumerate(values):
                self.alerts_table.setItem(row_index, column_index,
                    QTableWidgetItem(str(value)))

    def refresh_vehicles(self):
        connection = get_connection()
        cursor = connection.cursor()

```

```

rows = cursor.execute("""
    SELECT vehicle_id, vin, make, model, production_year, license_plate
    FROM vehicles
    ORDER BY vehicle_id
""").fetchall()

connection.close()

self.vehicles_table.setRowCount(len(rows))

for row_index, row in enumerate(rows):
    values = [
        row["vehicle_id"],
        row["vin"],
        row["make"],
        row["model"],
        row["production_year"],
        row["license_plate"]
    ]

    for column_index, value in enumerate(values):
        self.vehicles_table.setItem(row_index, column_index,
QTableWidgetItem(str(value)))

def apply_styles(app: QApplication):
    app.setStyleSheet("""
        QWidget {
            background-color: #f4f7fb;
            font-family: Arial;
            font-size: 14px;
            color: #1f2937;
        }

        QLineEdit {
            background-color: #ffffff;
            border: 1px solid #cbd5e1;
            border-radius: 6px;
            padding: 9px;
        }

        QPushButton {
            background-color: #1d4ed8;
            color: white;
            border: none;
            border-radius: 6px;
            padding: 10px 14px;
        }

        QPushButton:hover {
            background-color: #1e40af;
        }

        QTabWidget::pane {
            border: 1px solid #d1d5db;
            background-color: #ffffff;
        }

        QTableWidget {
            background-color: #ffffff;

```



```

        border: 1px solid #d1d5db;
        gridline-color: #e5e7eb;
    }

    QHeaderView::section {
        background-color: #e5edf7;
        padding: 8px;
        border: 1px solid #d1d5db;
    }

    QLabel#titleLabel {
        font-size: 18px;
        font-weight: 600;
        color: #1e3a8a;
    }

    QLabel#hintLabel {
        color: #6b7280;
        font-size: 12px;
    }

    QLabel#sectionTitle {
        font-size: 17px;
        font-weight: 600;
        color: #111827;
        padding: 8px;
    }

    QLabel#statCard {
        background-color: #ffffff;
        border: 1px solid #d1d5db;
        border-radius: 10px;
        padding: 18px;
        font-size: 16px;
    }
    """
)

def main():
    init_database()
    seed_database()

    app = QApplication(sys.argv)
    apply_styles(app)

    window = LoginWindow()
    window.show()

    sys.exit(app.exec())

if __name__ == "__main__":
    main()

```

ДОДАТОК Г**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ****Факультет інформаційних технологій****Декларація про академічну доброчесність та використання ШІ****ДЕКЛАРАЦІЯ ЗДОБУВАЧА**

Я, Возний Валентин Володимирович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення інформаційної системи обліку телеметрії автомобіля» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
 - ☐ інструменти генеративного ШІ не використовувалися.
 - ☐ інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____ інші (вказати назву)) були використані для:
 - ☐ перекладу іншомовних джерел;
 - ☐ стилістичного редагування та перевірки граматики;
 - ☐ допомоги у написанні/відлагодженні програмного коду;
 - ☐ інше: _____.

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

« ____ » _____ 2026 р. _____ Валентин ВОЗНИЙ
(підпис)