Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Комп'ютерних наук і технологій
(повне найменування факультету)

Комп'ютерні системи та мережі
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалаврський
(ступінь вищої освіти)

на тему: РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ МОНІТОРИНГУ СТАНУ АВТОМОБІЛЯ

Виконав(ла): студент(ка) 4 курсу,
групи КНТ-513сп

Спеціальності
123 Комп'ютерна інженерія
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

ЖМУРКО М.Д.
(ПРІЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.
(ПРІЗВИЩЕ та ініціали)
Рецензент РОМАНОВСЬКИЙ О.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
Кафедра комп'ютерних систем та мереж
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
_____
«_» ____2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЖМУРКА Максима Денисовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка комп'ютерної системи моніторингу стану автомобіля
керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ С.Ю.,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від « 14 » квітня 2026 року № 151
2. Строк подання студентом проєкту (роботи) 27.05.2026 р.
3. Вихідні дані до проєкту (роботи) мобільний Android застосунок системи моніторингу стану автомобіля, програмування Kotlin, фреймворк Jetpack Compose, серверна частина з використанням API, реалізація бази даних SQLite, клієнт-серверна архітектура, передача даних у форматі JSON.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області та засобів реалізації;
2) Проєктування системи;
3) Розробка комп'ютерної системи;
4) Випробування та демонстрація роботи системи.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5	СКРУПСЬКИЙ С.Ю.		
Нормоконтроль	ГОЛУБ Т.В.		

7. Дата видачі завдання «07» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Отримання завдання	до 07.04.2026	
2	Проведення огляду предметної області	до 10.04.2026	
3	Аналіз предметної області та засобів реалізації	до 19.04.2026	
4	Формування вимог до системи	до 24.04.2026	
5	Проєктування системи	до 28.05.2026	
6	Проєктування бази даних	до 01.05.2026	
7	Проєктування інтерфейсу користувача	до 05.05.2026	
8	Розробка комп'ютерної системи	до 10.05.2026	
9	Тестування та демонстрація роботи системи	до 15.05.2026	
10	Оформлення пояснювальної записки	до 20.05.2026	
11	Проходження нормоконтролю	до 29.05.2026	
12	Перевірка на наявність академічного плагіату	до 30.05.2026	
13	Проходження рецензування	до 31.05.2026	

Студент(ка) _____ Максим ЖМУРКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 116 с., 9 табл., 52 рис., 2 дод., 40 джерел.

МОНІТОРИНГ СТАНУ АВТОМОБІЛЯ, МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, KOTLIN, JETPACK COMPOSE, API, КЛІЄНТ-СЕРВЕР, TELEMETRY, SQLITE

Розроблена комп'ютерна система моніторингу стану автомобіля містить клієнтську та серверну частини.

Клієнтська частина реалізована у вигляді мобільного застосунку для операційної системи Android та забезпечує взаємодію користувача із системою. Мобільний застосунок дозволяє виконувати авторизацію користувача, переглядати телеметричні параметри автомобіля, отримувати інформацію про стан транспортного засобу та переглядати статистичні дані.

Серверна частина забезпечує обробку запитів від мобільного застосунку, передачу телеметричних параметрів та взаємодію із базою даних системи. Передача інформації між компонентами програмного забезпечення здійснюється за допомогою API-запитів у форматі JSON.

У процесі виконання дипломної роботи було реалізовано клієнт-серверну архітектуру системи, структуру бази даних та механізми обробки телеметричних параметрів автомобіля. Для реалізації мобільного застосунку використовувались Android Studio, мова програмування Kotlin та фреймворк Jetpack Compose.

Архітектура клієнт-сервер, яка використовується у даній роботі, забезпечує взаємодію між окремими компонентами системи та дозволяє реалізувати мобільний застосунок для моніторингу стану автомобіля у режимі реального часу.

ЗМІСТ

ВСТУП

У сучасних автомобілях активно використовуються електронні системи керування, цифрові датчики та програмні засоби контролю технічного стану транспортного засобу. Це дозволяє отримувати інформацію про роботу двигуна, температуру, витрату пального, швидкість руху, наявність помилок та інші телеметричні параметри. Використання подібних даних дає можливість своєчасно виявляти несправності, контролювати стан автомобіля та підвищувати безпеку його експлуатації.

Особливої актуальності набувають комп’ютерні системи моніторингу автомобіля, які дозволяють у режимі реального часу отримувати та аналізувати інформацію про стан транспортного засобу за допомогою мобільних застосунків. Такі системи широко використовуються як приватними користувачами, так і сервісними центрами, логістичними компаніями та підприємствами, діяльність яких пов’язана з транспортом.

Незважаючи на наявність великої кількості готових програмних рішень для автомобільної діагностики, значна частина існуючих систем має обмежений функціонал, не забезпечує повноцінного збереження телеметричних даних або потребує використання платних сервісів. Крім того, багато застосунків орієнтовані лише на зчитування помилок автомобіля без можливості побудови повноцінної клієнт-серверної системи моніторингу.

Дана дипломна робота присвячена розробці комп’ютерної системи моніторингу стану автомобіля на основі клієнт-серверної архітектури. Основною метою проєкту є створення мобільного застосунку для отримання, збереження та відображення телеметричних даних транспортного засобу.

У першому розділі проведено аналіз предметної області, розглянуто особливості сучасних систем моніторингу автомобілів, виконано огляд існуючих програмних рішень та визначено основні вимоги до майбутньої системи.

Другий розділ присвячено проєктуванню комп’ютерної системи моніторингу стану автомобіля. У розділі сформовано функціональні вимоги до програмного забезпечення, розроблено архітектуру системи, діаграму варіантів використання, структуру бази даних та схему збору і передачі телеметричних даних через інтерфейс OBD-II.

У третьому розділі описано процес реалізації мобільного застосунку, серверної частини та бази даних системи. Наведено приклади програмного коду, реалізацію взаємодії із сервером, механізми отримання телеметричних параметрів через адаптер ELM327 та особливості побудови користувацького інтерфейсу.

Четвертий розділ присвячено випробуванню розробленої системи. У межах розділу виконано перевірку отримання телеметричних даних, передачі інформації між компонентами системи, збереження даних у базі даних та перевірку основних сценаріїв роботи користувача із програмним забезпеченням.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАСОБІВ РЕАЛІЗАЦІЇ

1.1 Сучасні системи моніторингу стану автомобіля

Сучасні автомобілі містять велику кількість електронних систем та датчиків, які забезпечують контроль технічного стану транспортного засобу. Контроль параметрів автомобіля дозволяє своєчасно виявляти несправності, аналізувати роботу окремих систем та підвищувати безпеку експлуатації транспортного засобу [1].

Одним із напрямів розвитку автомобільних інформаційних систем є створення програмних засобів для моніторингу технічного стану автомобіля у режимі реального часу. Подібні системи дозволяють отримувати інформацію про роботу двигуна, температуру охолоджувальної рідини, рівень пального, швидкість руху, стан акумулятора та інші параметри [2].

Сучасні системи моніторингу стану автомобіля можуть реалізовуватись у різних формах залежно від особливостей використання та технічних можливостей транспортного засобу. Найбільш поширеними є мобільні застосунки, які забезпечують швидкий доступ до основних параметрів автомобіля за допомогою смартфона. Окрім мобільних рішень, подібні системи можуть функціонувати у вигляді веб платформ із можливістю віддаленого доступу через браузер, вбудованих мультимедійних систем автомобіля або хмарних сервісів, що забезпечують централізоване збереження телеметричних даних та синхронізацію інформації між декількома пристроями [3].

Більшість сучасних рішень використовують клієнт-серверну архітектуру, у межах якої мобільний застосунок або веб інтерфейс виступає клієнтською частиною системи, а сервер забезпечує збереження, обробку та синхронізацію даних [4]. Такий підхід дозволяє організувати централізовану обробку інформації та забезпечує можливість віддаленого доступу до параметрів транспортного засобу незалежно від місця перебування користувача.

Використання систем моніторингу стану автомобіля дозволяє значно спростити контроль технічних параметрів транспортного засобу та підвищити зручність його експлуатації. Завдяки автоматичному збору та аналізу інформації користувач отримує можливість оперативно контролювати стан автомобіля, переглядати історію показників та своєчасно реагувати на появу несправностей [5].

Додатковою перевагою є можливість формування повідомлень про критичні зміни параметрів, що дозволяє підвищити безпеку використання транспортного засобу та зменшити ризик виникнення серйозних технічних проблем.

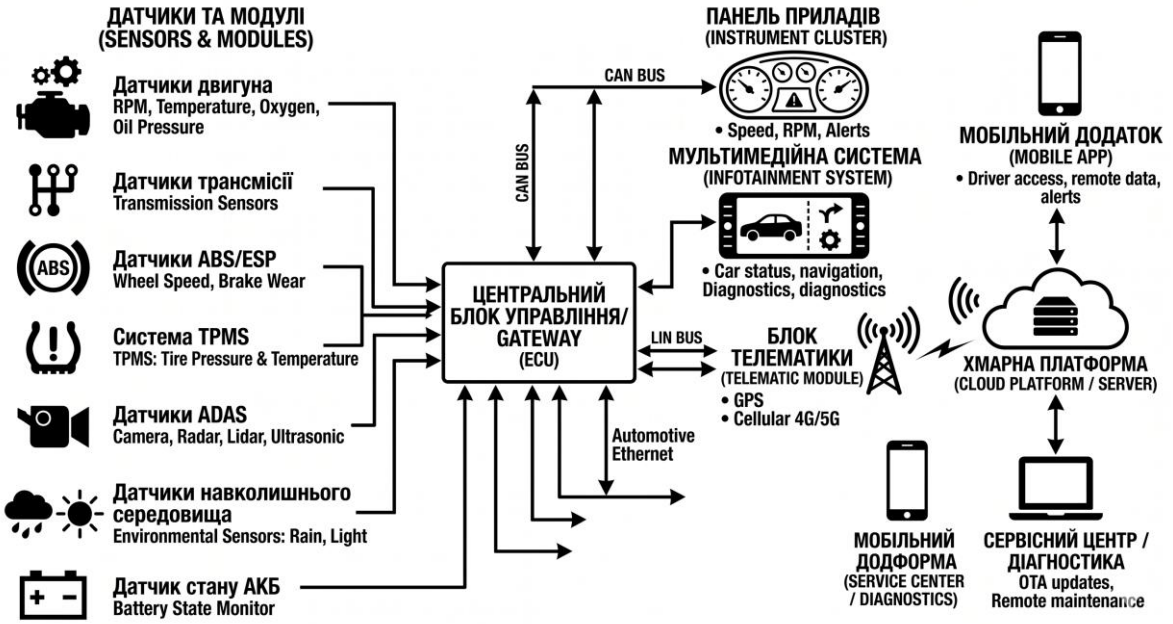

Рисунок 1.1 - Приклад структури сучасної системи моніторингу автомобіля

1.2 Основні параметри моніторингу транспортного засобу

Ефективність системи моніторингу безпосередньо залежить від набору параметрів, які отримуються та аналізуються під час роботи автомобіля. Контроль технічних показників дозволяє оцінювати стан окремих систем транспортного засобу та своєчасно реагувати на можливі несправності [6].

До основних параметрів, що можуть використовуватись у системах моніторингу автомобіля, належать:

- швидкість руху;
- температура двигуна;
- рівень пального;
- напруга акумулятора;
- витрата пального;
- оберти двигуна;
- пробіг автомобіля;
- коди помилок електронних систем.

Частина даних може отримуватись через електронні системи автомобіля та інтерфейс OBD-II. Використання OBD-II дозволяє зчитувати інформацію із електронного блоку керування автомобіля та передавати її до клієнтського застосунку [7].

Таблиця 1.1 - Основні параметри моніторингу автомобіля

Параметр	Призначення
Температура двигуна	Контроль перегріву двигуна
Рівень пального	Аналіз залишку пального
Напруга акумулятора	Контроль стану електроживлення
Оберти двигуна	Аналіз роботи двигуна
Швидкість руху	Моніторинг режиму експлуатації
Коди помилок	Виявлення несправностей

Контроль зазначених параметрів дозволяє підвищити надійність експлуатації транспортного засобу та спростити процес технічного обслуговування.

1.3 Використання клієнт-серверної архітектури

Одним із найбільш поширених підходів до побудови сучасних інформаційних систем є використання клієнт-серверної архітектури [8]. Такий підхід дозволяє розділити систему на окремі компоненти, що відповідають за обробку даних, збереження інформації та взаємодію із користувачем.

У межах системи моніторингу стану автомобіля клієнтська частина може бути реалізована у вигляді мобільного застосунку, який забезпечує взаємодію користувача із системою та відображення основних параметрів транспортного засобу. За допомогою мобільного застосунку користувач отримує можливість проходити авторизацію, переглядати поточний стан автомобіля, контролювати історію отриманих показників та отримувати повідомлення про виявлені помилки або критичні зміни параметрів [9].

Серверна частина системи відповідає за обробку запитів користувачів, збереження інформації та синхронізацію даних між окремими компонентами системи. На сервері може здійснюватися авторизація користувачів, формування статистики роботи автомобіля, обробка телеметричних даних та збереження історії показників [10].

Використання клієнт-серверної архітектури дозволяє централізовано зберігати інформацію та забезпечує можливість доступу до даних із різних пристроїв.

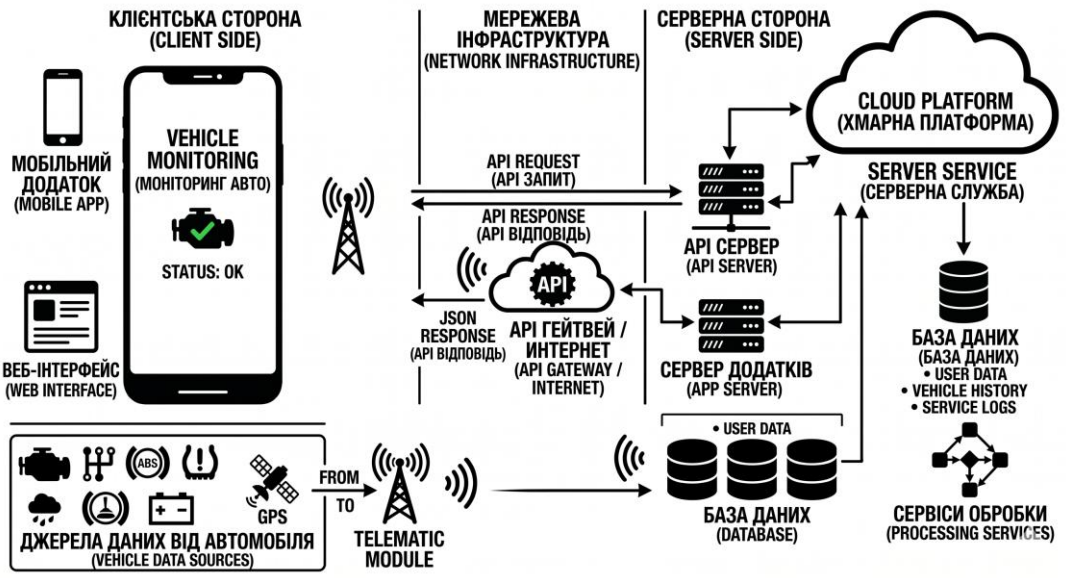

Рисунок 1.2 - Умовна загальна схема клієнт-серверної системи

1.4 Вибір технологій для реалізації системи

Під час проектування системи моніторингу стану автомобіля важливим етапом є вибір технологій, які будуть використовуватись для реалізації клієнтської та серверної частин системи [11].

Для створення мобільного застосунку можуть використовуватись сучасні фреймворки та платформи розробки, які забезпечують підтримку адаптивного інтерфейсу та взаємодію із сервером через API [12].

Серверна частина системи повинна забезпечувати:

- стабільну обробку запитів;
- збереження даних;
- підтримку авторизації;
- роботу із базою даних;
- передачу телеметричної інформації.

Для збереження інформації доцільним є використання реляційної бази даних, яка забезпечує структуроване зберігання інформації та підтримує ефективний пошук даних [13].

Таблиця 1.2 - Основні технології системи

Технологія	Призначення
Температура двигуна	Контроль перегріву двигуна
Рівень пального	Аналіз залишку пального
Напруга акумулятора	Контроль стану електроживлення
Оберти двигуна	Аналіз роботи двигуна
Швидкість руху	Моніторинг режиму експлуатації
Коди помилок	Виявлення несправностей

1.5 Формування функціональних вимог до системи

Функціональні вимоги визначають основні можливості системи та перелік задач, які вона повинна виконувати під час роботи.

Система моніторингу стану автомобіля повинна забезпечувати:

- авторизацію користувача;
- додавання автомобіля до системи;
- отримання телеметричних даних;
- перегляд поточних параметрів автомобіля;
- збереження історії показників;
- формування повідомлень про критичні стани;
- перегляд статистики роботи автомобіля.

Користувацький інтерфейс повинен забезпечувати швидкий доступ до основної інформації про транспортний засіб та підтримувати зручну навігацію між екранами застосунку [14].

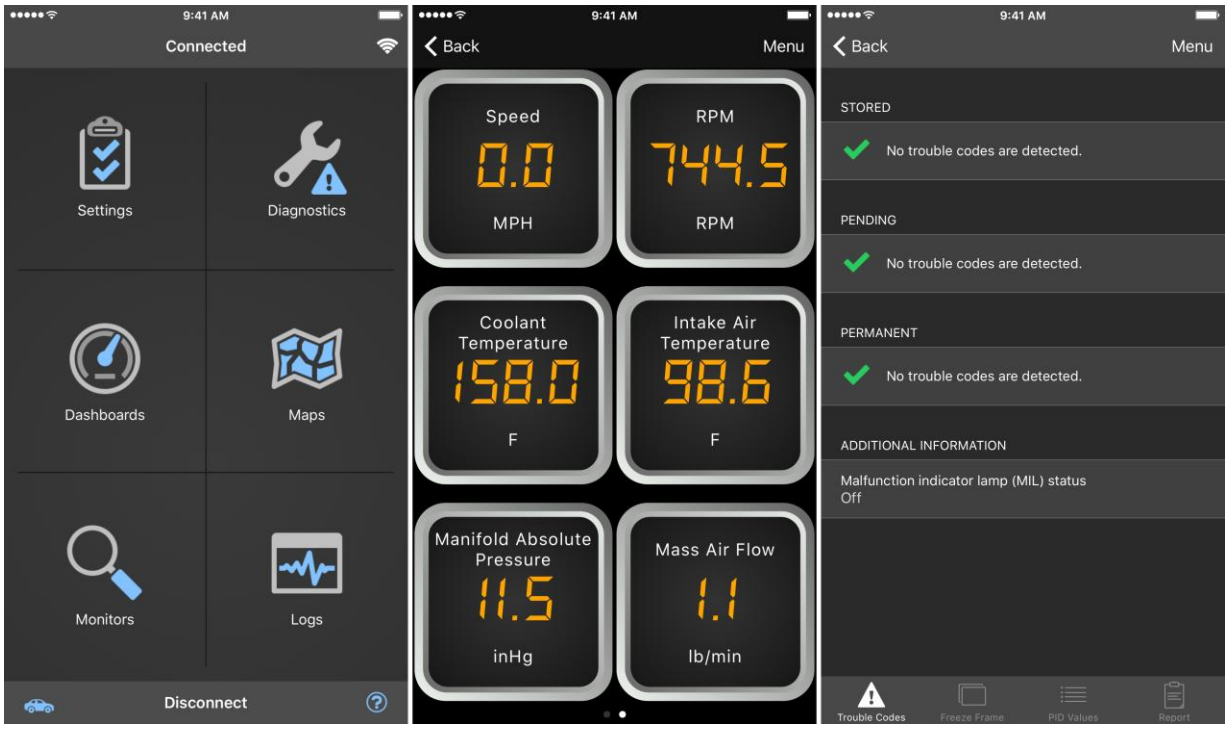

Рисунок 1.3 - Приклад структури інтерфейсу мобільного застосунку

1.6 Формування нефункціональних вимог

Окрім функціональних можливостей, система повинна відповідати нефункціональним вимогам, які визначають загальні характеристики програмного забезпечення [15].

До основних нефункціональних вимог системи належать:

- стабільність роботи;
- швидкість обробки запитів;
- зручність інтерфейсу користувача;
- масштабованість;
- безпечне збереження даних;
- підтримка мобільних пристроїв;
- можливість подальшого розширення функціоналу.

Важливим аспектом є забезпечення коректної роботи системи при нестабільному мережевому з’єднанні та підтримка синхронізації даних між клієнтом і сервером [16].

Під час проектування інтерфейсу необхідно враховувати особливості використання мобільних пристроїв, включаючи різні розміри екранів та сенсорне керування.

Таблиця 1.3 - Нефункціональні вимоги до системи

Вимога	Характеристика
Надійність	Стабільна робота системи
Масштабованість	Можливість розширення функціоналу
Адаптивність	Підтримка різних пристроїв
Зручність	Зручність функціоналу для користувачів
Продуктивність	Швидка обробка запитів

1.7 Висновки до розділу

У результаті проведеного аналізу предметної області було досліджено особливості сучасних систем моніторингу стану автомобілів та визначено основні принципи побудови комп’ютерної системи, призначеної для збору, обробки та відображення телеметричної інформації. Встановлено, що розвиток електронних систем керування транспортними засобами створює можливість отримання значної кількості технічних параметрів у режимі реального часу, що дозволяє підвищити ефективність контролю технічного стану автомобіля.

Аналіз способів отримання телеметричних даних показав, що найбільш поширеним рішенням є використання інтерфейсу OBD-II, який забезпечує доступ

до інформації, що формується електронними блоками керування автомобіля. Використання даного інтерфейсу дозволяє отримувати параметри роботи двигуна, швидкість руху транспортного засобу, температуру охолоджувальної рідини, оберти двигуна та інші показники, необхідні для моніторингу технічного стану автомобіля. Саме тому технологія OBD-II була обрана як основне джерело даних для розроблюваної системи.

Дослідження існуючих програмних рішень показало, що більшість сучасних систем поєднують функції діагностики, збору телеметричних даних та відображення інформації у мобільних застосунках. Разом з тим значна частина доступних рішень є орієнтованою на конкретних виробників автомобілів або містить обмеження щодо функціональних можливостей без використання платних сервісів. Це підтверджує доцільність розробки власної системи моніторингу, яка забезпечуватиме базові функції контролю технічного стану транспортного засобу незалежно від марки автомобіля.

На основі проведеного аналізу було сформовано функціональні вимоги до системи. Визначено необхідність реалізації механізмів авторизації користувачів, додавання транспортних засобів, отримання телеметричних даних, перегляду поточних параметрів автомобіля та збереження історії показників. Окрему увагу приділено забезпеченню зручного відображення інформації та можливості подальшого розширення функціональних можливостей системи.

Таким чином, результати першого розділу дозволили сформувати загальне уявлення про предметну область, визначити основні джерела отримання даних та обґрунтувати вибір підходів, які були використані під час проєктування і реалізації комп’ютерної системи моніторингу стану автомобіля.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Загальна структура системи

Під час розробки комп’ютерної системи моніторингу стану автомобіля важливим етапом є проєктування загальної структури системи та принципу взаємодії її основних компонентів. Правильно спроєктована структура програмного забезпечення дозволяє забезпечити стабільну роботу системи, спростити процес обробки інформації та реалізувати зручну взаємодію користувача із функціональними можливостями застосунку [17].

Для реалізації системи моніторингу стану автомобіля було обрано клієнт-серверну архітектуру. Подібний підхід широко використовується під час розробки сучасних мобільних застосунків та інформаційних систем, оскільки дозволяє розділити функції між окремими компонентами системи та забезпечити централізоване збереження інформації [18].

У межах розробленої системи мобільний застосунок виступає клієнтською частиною та забезпечує взаємодію користувача із системою. Основним призначенням клієнтської частини є відображення інформації про стан автомобіля, отримання телеметричних даних, виконання авторизації користувача та передача запитів до серверної частини системи.

Серверна частина відповідає за обробку запитів користувача, роботу з базою даних та передачу інформації між окремими компонентами системи. На сервері здійснюється збереження інформації про користувачів, транспортні засоби та телеметричні параметри автомобіля [19].

Для централізованого збереження інформації у системі використовується база даних, яка забезпечує збереження історії показників, інформації про користувачів та параметрів транспортних засобів. Використання бази даних дозволяє реалізувати механізм накопичення телеметричних даних та подальший перегляд історії параметрів автомобіля [20].

Загальна структура системи включає наступні компоненти:

– мобільний застосунок;

– серверну частину;

– базу даних;

– API взаємодії між компонентами;

– систему отримання телеметричних даних.

Під час роботи системи користувач проходить авторизацію у мобільному застосунку, після чого отримує доступ до основного функціоналу системи. Клієнтська частина надсилає запити до сервера за допомогою API, а сервер обробляє отриману інформацію та взаємодіє із базою даних. Після виконання запиту сервер повертає результат до мобільного застосунку у форматі JSON [21].

Важливим компонентом системи є механізм отримання телеметричних даних автомобіля. Для отримання інформації про параметри транспортного засобу використовується інтерфейс OBD-II, який дозволяє отримувати інформацію про роботу двигуна, температуру охолоджувальної рідини, рівень пального, швидкість руху та інші параметри автомобіля [7].

Використання клієнт-серверної архітектури забезпечує централізоване збереження інформації, можливість доступу до системи із різних пристроїв та спрощує процес подальшого розширення функціоналу системи [18].

Рисунок 2.1 - Схема структури системи моніторингу автомобіля

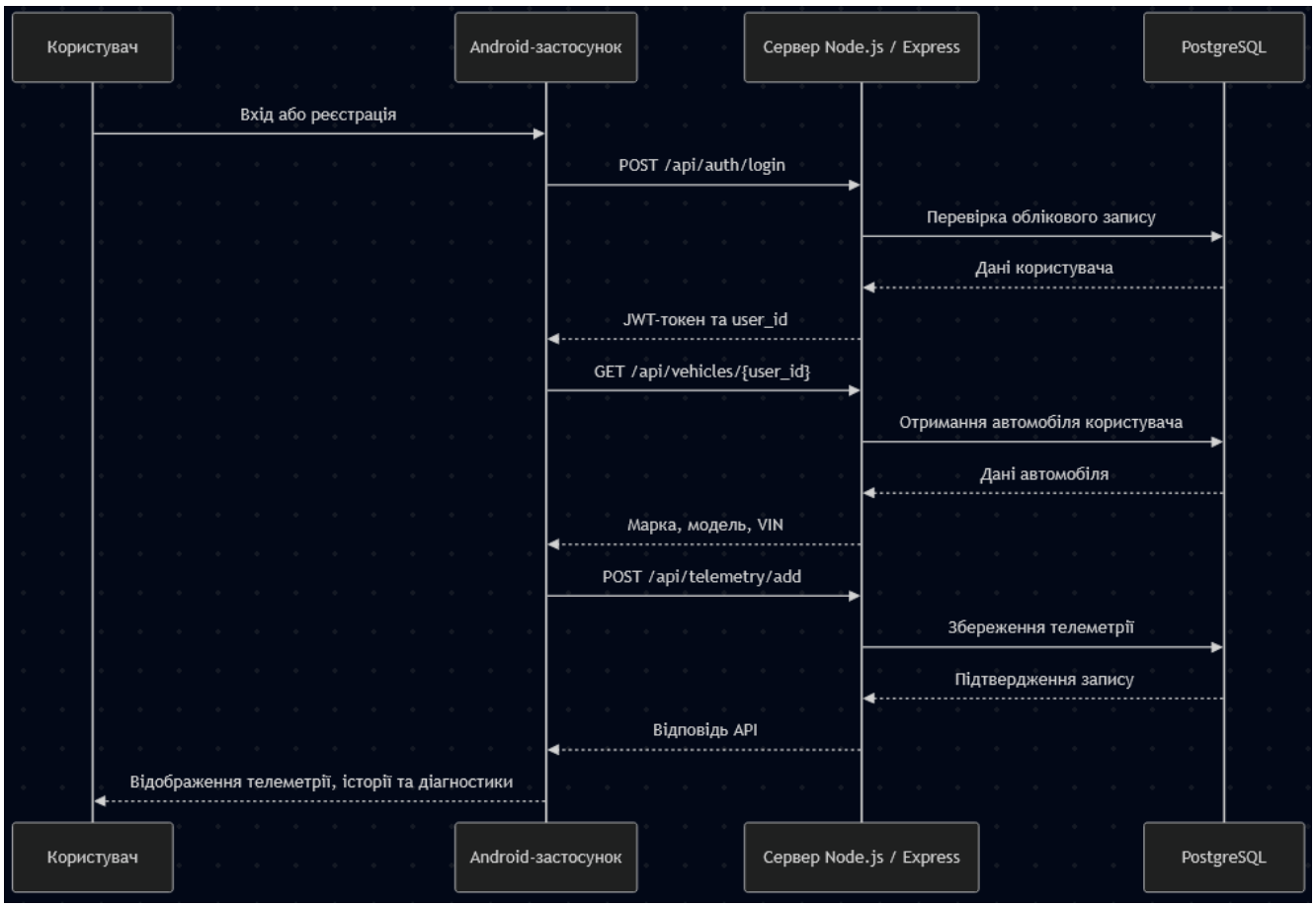

Рисунок 2.2 - Схема взаємодії компонентів системи

2.2 Функціональні можливості системи

Одним із основних етапів проєктування комп’ютерної системи моніторингу стану автомобіля є визначення функціональних можливостей системи. Формування функціональних вимог дозволяє визначити перелік задач, які повинна виконувати система під час роботи, а також встановити основні можливості взаємодії користувача із програмним забезпеченням [22].

Розроблена система моніторингу стану автомобіля орієнтована на забезпечення зручного доступу до інформації про транспортний засіб та отримання основних телеметричних параметрів автомобіля у режимі реального часу. Основний функціонал системи реалізовано у вигляді мобільного застосунку, який забезпечує взаємодію користувача із серверною частиною та базою даних.

Однією із ключових функцій системи є авторизація користувача. Використання механізму авторизації дозволяє забезпечити захист персональної інформації та розмежування доступу до даних окремих користувачів. Після проходження авторизації користувач отримує доступ до функціоналу системи та можливість роботи із власними транспортними засобами [23].

Система забезпечує можливість додавання автомобіля до облікового запису користувача. Після додавання транспортного засобу користувач отримує можливість переглядати інформацію про автомобіль та здійснювати моніторинг його основних параметрів.

Однією із основних функцій системи є отримання телеметричних даних автомобіля. Для цього використовуються електронні системи транспортного засобу та інтерфейс OBD-II, який забезпечує передачу інформації про технічний стан автомобіля [7]. Отримані параметри можуть передаватись до серверної частини системи для подальшого збереження та аналізу.

Система забезпечує можливість перегляду поточних параметрів автомобіля. Користувач може отримувати інформацію про швидкість руху, температуру двигуна, рівень пального, оберти двигуна, стан акумулятора та інші показники транспортного засобу [24].

Для забезпечення можливості аналізу роботи автомобіля у системі реалізовано механізм збереження історії телеметричних показників. Це дозволяє переглядати попередні параметри транспортного засобу та аналізувати зміни технічного стану автомобіля у різні періоди часу.

Окремою функціональною можливістю системи є формування повідомлень про критичні стани автомобіля. У випадку виявлення помилок або перевищення допустимих значень окремих параметрів система може повідомляти користувача про можливу несправність транспортного засобу [25].

Також система забезпечує можливість перегляду статистики роботи автомобіля. Статистична інформація може використовуватись для аналізу загального стану транспортного засобу та контролю окремих показників роботи автомобіля.

Основні функціональні можливості системи наведено нижче:

– авторизація користувача;

– додавання автомобіля до системи;

– отримання телеметричних даних;

– перегляд поточних параметрів автомобіля;

– збереження історії показників;

– формування повідомлень про критичні стани;

– перегляд статистики роботи автомобіля.

Під час проєктування функціональних можливостей системи також було враховано необхідність забезпечення простого та зрозумілого користувацького інтерфейсу. Навігація між екранами мобільного застосунку повинна забезпечувати швидкий доступ до основних функцій системи та спрощувати процес взаємодії користувача із програмним забезпеченням [26].

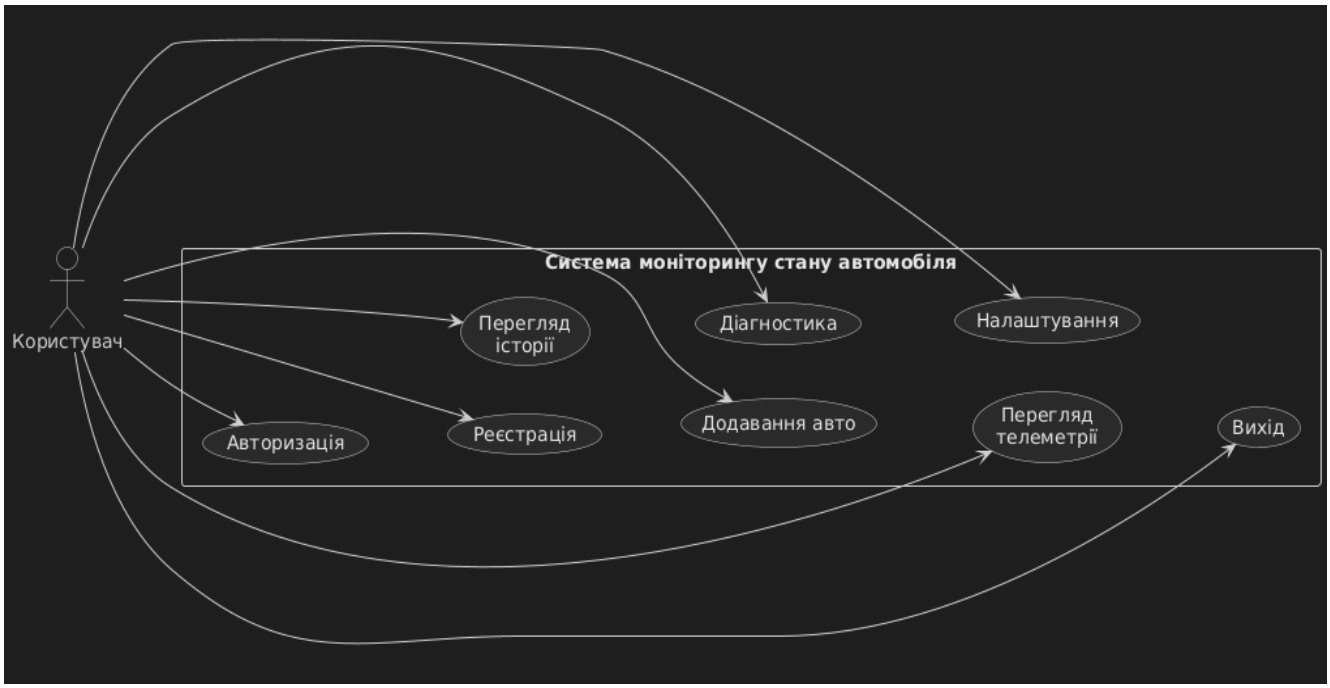

Рисунок 2.3 - Основні функціональні можливості системи

2.3 Організація збору телеметричних даних автомобіля

Однією з основних функцій комп’ютерної системи моніторингу стану автомобіля є отримання телеметричних даних від транспортного засобу. Саме телеметрична інформація дозволяє оцінювати поточний технічний стан автомобіля, контролювати роботу його основних вузлів та своєчасно виявляти можливі відхилення від нормального режиму роботи. Використання телеметричних систем є одним із найбільш поширених підходів у сучасних рішеннях для моніторингу транспортних засобів та керування автопарками [2].

Джерелом інформації про стан автомобіля виступає електронний блок керування (ECU), який здійснює контроль роботи двигуна та інших електронних систем транспортного засобу. Дані про роботу автомобіля формуються на основі інформації, отриманої від численних датчиків, після чого можуть бути передані через стандартний діагностичний інтерфейс OBD-II (On-Board Diagnostics II) [6].

Стандарт OBD-II використовується у більшості сучасних автомобілів та забезпечує уніфікований доступ до діагностичної інформації незалежно від виробника транспортного засобу. Використання даного інтерфейсу дозволяє отримувати інформацію про швидкість руху автомобіля, оберти двигуна, температуру охолоджувальної рідини, навантаження на двигун, напругу бортової мережі та інші параметри, необхідні для моніторингу технічного стану транспортного засобу [6]. Важливою перевагою використання OBD-II є стандартизація процесу обміну даними, що забезпечує сумісність діагностичного обладнання із великою кількістю моделей автомобілів та спрощує інтеграцію засобів моніторингу у програмні системи різного призначення [29].

Для зчитування інформації із діагностичного роз'єму використовується адаптер ELM327, який підключається до порту OBD-II та виконує роль посередника між автомобілем і програмним забезпеченням. Адаптер здійснює обмін даними з електронним блоком керування та забезпечує передачу отриманої інформації до мобільного застосунку через інтерфейс Bluetooth. Такий підхід

дозволяє організувати збір телеметричних даних без внесення змін у конструкцію автомобіля та без використання спеціалізованого сервісного обладнання [29]. Крім того, використання бездротового з'єднання підвищує зручність експлуатації системи та дозволяє здійснювати моніторинг параметрів транспортного засобу безпосередньо під час його використання.

Отримання інформації виконується за допомогою спеціальних PID-команд (Parameter IDs), які використовуються для запиту окремих параметрів транспортного засобу. Кожний PID відповідає певному показнику автомобіля та дозволяє отримувати значення у стандартизованому форматі [7]. Завдяки використанню PID-команд забезпечується можливість отримання необхідних параметрів незалежно від конкретної моделі автомобіля, що значно спрощує реалізацію систем моніторингу та діагностики. Отримані значення можуть використовуватися не лише для поточного відображення інформації користувачу, але й для накопичення статистики, аналізу зміни параметрів у часі та виявлення потенційних несправностей на ранніх етапах їх виникнення [5].

Після отримання телеметричних даних мобільний застосунок виконує їх обробку та передає інформацію до серверної частини системи. Сервер забезпечує централізоване збереження отриманих показників у базі даних PostgreSQL та надає можливість подальшого доступу до історії телеметричних параметрів. Така схема дозволяє реалізувати повний цикл обробки інформації — від отримання даних із автомобіля до їх відображення користувачу у мобільному застосунку та збереження для подальшого аналізу [5].

Загальну схему збору та передачі телеметричних даних наведено на рисунку 2.4.

Рисунок 2.4 - Схема збору та передачі телеметричних даних

Важливу роль у процесі отримання телеметричної інформації відіграють PID-команди (Parameter IDs), які використовуються для запиту окремих параметрів автомобіля через інтерфейс OBD-II. Кожна команда відповідає певному параметру транспортного засобу та дозволяє отримувати значення у стандартизованому форматі незалежно від виробника автомобіля [7].

Під час роботи системи мобільний застосунок або адаптер ELM327 формує запит до електронного блоку керування автомобіля. Після отримання запиту ECU

повертає відповідне значення параметра, яке надалі може бути оброблене та відображене користувачу. Використання стандартизованих PID-команд забезпечує сумісність системи з великою кількістю транспортних засобів та спрощує реалізацію механізму збору телеметричних даних [6].

У межах розробленої системи для моніторингу стану автомобіля використовуються команди отримання основних параметрів, які характеризують роботу двигуна та загальний технічний стан транспортного засобу. До таких параметрів належать швидкість руху автомобіля, оберти двигуна, температура охолоджувальної рідини та напруга бортової мережі. Отримана інформація використовується для відображення поточного стану автомобіля у мобільному застосунку та подальшого збереження у базі даних для аналізу історії показників [29].

Основні PID-команди, які можуть використовуватись для отримання телеметричних параметрів автомобіля, наведені у таблиці 2.1.

Таблиця 2.1 - Основні PID-команди OBD-II

Параметр	PID	Опис
Швидкість автомобіля	010D	Поточна швидкість руху транспортного засобу
Оберти двигуна	010C	Кількість обертів колінчастого вала двигуна
Температура двигуна	0105	Температура охолоджувальної рідини
Напруга бортової мережі	0145	Напруга електричної системи автомобіля

2.4 Діаграма варіантів використання системи

Під час проєктування комп’ютерної системи моніторингу стану автомобіля важливим етапом є визначення основних сценаріїв взаємодії користувача із програмним забезпеченням. Для цього використовується діаграма варіантів

використання (Use Case Diagram), яка дозволяє відобразити базові функціональні можливості системи та взаємодію користувача із окремими компонентами застосунку [27].

Діаграма варіантів використання використовується для візуального представлення функціоналу системи та демонструє набір дій, які може виконувати користувач під час роботи із програмним забезпеченням. Подібний підхід дозволяє спростити процес проєктування системи та визначити основні функціональні можливості ще до початку реалізації програмного забезпечення [28].

У розробленій системі основним актором виступає користувач мобільного застосунку. Саме користувач взаємодіє із клієнтською частиною системи, отримує інформацію про транспортний засіб та виконує основні дії, пов’язані із моніторингом стану автомобіля.

До основних варіантів використання системи належать:

– авторизація користувача;

– додавання автомобіля до системи;

– перегляд інформації про автомобіль;

– отримання телеметричних даних;

– перегляд історії показників;

– перегляд статистики роботи автомобіля;

– отримання повідомлень про помилки та критичні стани.

Одним із основних сценаріїв роботи системи є авторизація користувача. Після успішного входу до системи користувач отримує доступ до основного функціоналу мобільного застосунку та можливість роботи із власними транспортними засобами [23].

Важливим сценарієм є додавання автомобіля до системи. Після виконання цієї дії користувач отримує можливість переглядати інформацію про транспортний засіб та здійснювати моніторинг його параметрів.

Окремим варіантом використання є отримання телеметричних даних автомобіля. У процесі роботи система отримує інформацію про стан

транспортного засобу через інтерфейс OBD-II та відображає її у мобільному застосунку [7].

Також користувач отримує можливість переглядати історію показників та статистику роботи автомобіля. Функціонал дозволяє аналізувати зміни параметрів транспортного засобу та контролювати технічний стан автомобіля протягом певного періоду часу [29].

У випадку виникнення помилок або критичних змін параметрів система формує відповідні повідомлення для користувача. Це дозволяє своєчасно реагувати на можливі несправності транспортного засобу та підвищує безпеку експлуатації автомобіля [25].

Під час побудови діаграми варіантів використання було враховано лише основний функціонал системи. Для побудови діаграми варіантів використання використовуються основні елементи UML-моделювання, опис яких наведено у таблиці 2.2.

Таблиця 2.2 - Основні елементи діаграми варіантів використання

Назва елементу	Призначення
Актор	Представляє користувача або зовнішній компонент, який взаємодіє із системою
Варіант використання	Відображає окрему функціональну можливість системи
Зв’язок	Використовується для відображення взаємодії між актором та функціоналом
Межа системи	Визначає межі функціонування програмної системи

2.5 Проєктування бази даних системи

Одним із основних етапів проєктування комп’ютерної системи моніторингу стану автомобіля є розробка структури бази даних. База даних використовується для централізованого збереження інформації про користувачів, транспортні засоби та телеметричні параметри автомобіля. Використання структурованого сховища даних дозволяє забезпечити цілісність інформації, спростити процес обробки даних та реалізувати механізм збереження історії показників [30].

Під час проєктування структури бази даних було враховано функціональні можливості системи, особливості взаємодії між окремими компонентами та необхідність збереження телеметричної інформації. Основними сутностями бази даних є користувачі, автомобілі та телеметричні дані. Такий підхід дозволяє логічно розділити інформацію за її призначенням, забезпечити зручність подальшої обробки даних та спростити реалізацію зв’язків між окремими елементами системи.

Для побудови структури бази даних було використано ER-модель, яка дозволяє візуально відобразити зв’язки між окремими таблицями та структуру збереження інформації [31]. Використання ER-моделювання дає можливість ще на етапі проєктування визначити основні сутності предметної області, встановити зв’язки між ними та оцінити ефективність обраної структури даних. Крім того, така модель спрощує подальшу реалізацію бази даних та її інтеграцію із серверною частиною системи.

На рисунку 2.5 наведено ER-модель бази даних системи моніторингу стану автомобіля.

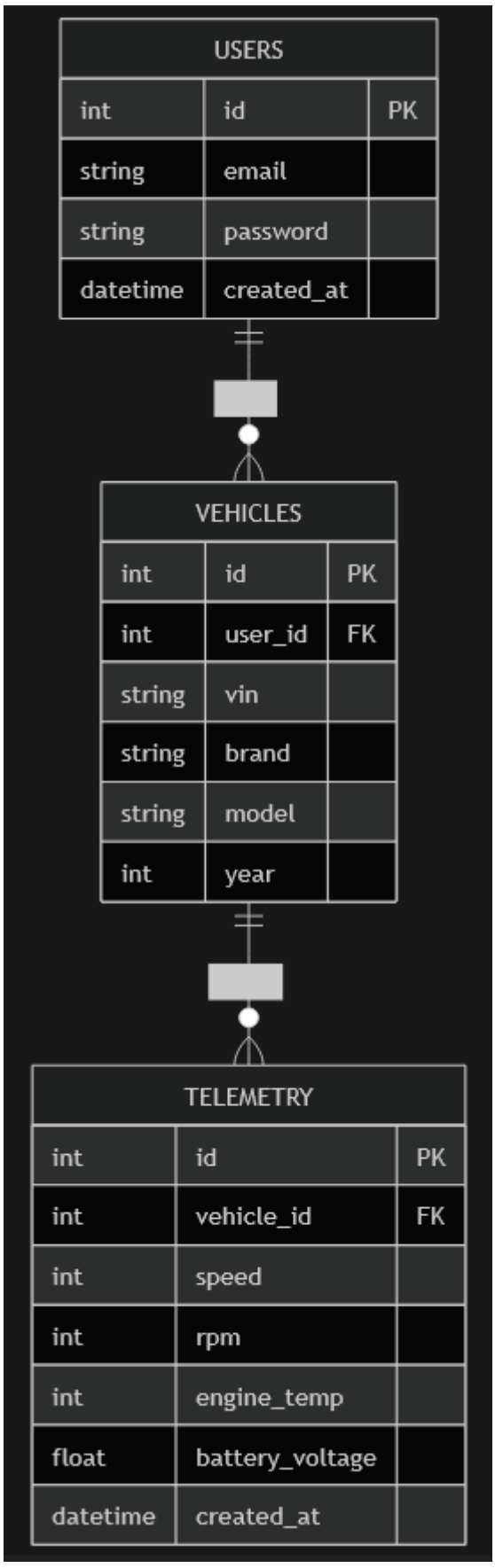

Рисунок 2.5 - ER-модель бази даних системи

Однією із основних таблиць бази даних є таблиця користувачів. У даній таблиці зберігається інформація про зареєстрованих користувачів системи, включаючи логін, адресу електронної пошти та пароль користувача.

Структуру таблиці користувачів наведено у таблиці 2.3.

Таблиця 2.3 - Структура таблиці користувачів

Атрибут	Тип даних	Призначення
user_id	INTEGER	Унікальний ідентифікатор користувача
email	TEXT	Електронна пошта
password	TEXT	Пароль користувача

Наступною важливою таблицею є таблиця автомобілів. У даній таблиці зберігається інформація про транспортні засоби, які були додані користувачами до системи.

Таблиця автомобілів містить інформацію про модель автомобіля, рік випуску, ідентифікаційні дані транспортного засобу та зв’язок із конкретним користувачем системи.

Структуру таблиці автомобілів наведено у таблиці 2.4.

Таблиця 2.4 - Структура таблиці автомобілів

Атрибут	Тип даних	Призначення
car_id	INTEGER	Унікальний ідентифікатор автомобіля
user_id	INTEGER	Унікальний ідентифікатор користувача
brand	TEXT	Марка автомобіля
model	TEXT	Модель автомобіля
year	TEXT	Рік випуску
vin	TEXT	VIN-код автомобіля

Для збереження телеметричних параметрів використовується окрема таблиця телеметричних даних. У даній таблиці зберігається інформація про

технічний стан автомобіля та показники, отримані під час роботи системи моніторингу [32].

До основних параметрів, що зберігаються у таблиці телеметричних даних, належать:

– швидкість руху;

– температура двигуна;

– оберти двигуна;

– напруга акумулятора;

– дата та час отримання інформації.

Структуру таблиці телеметричних даних наведено у таблиці 2.5.

Таблиця 2.5 - Структура таблиці телеметричних даних

Атрибут	Тип даних	Призначення
car_id	INTEGER	Унікальний ідентифікатор автомобіля
telemetry_id	INTEGER	Унікальний ідентифікатор запису
speed	REAL	Швидкість руху
engine_temp	REAL	Температура двигуна
rpm	REAL	Оберти двигуна
battery_voltage	REAL	Напруга акумулятора
created_at	DATETIME	Дата та час запису

Між таблицями бази даних реалізовано зв’язки типу «один до багатьох». Один користувач може мати декілька автомобілів, а один автомобіль може містити велику кількість телеметричних записів [31].

Під час проєктування бази даних також було враховано необхідність забезпечення швидкого доступу до інформації та можливість подальшого розширення структури системи. Використання окремих таблиць для користувачів, автомобілів та телеметричних даних дозволяє спростити процес збереження інформації та забезпечує зручну взаємодію між окремими компонентами системи [30].

Окрім цього, структура бази даних дозволяє у майбутньому додавати нові параметри моніторингу або додаткові функціональні можливості без необхідності повної зміни структури системи.

2.6 Проєктування структури мобільного застосунку

Під час проєктування комп’ютерної системи моніторингу стану автомобіля значна увага приділялась структурі мобільного застосунку та організації користувацького інтерфейсу. Основним призначенням мобільного застосунку є забезпечення взаємодії користувача із системою, перегляд параметрів автомобіля та отримання телеметричних даних у зручному вигляді [33].

Мобільний застосунок було спроєктовано таким чином, щоб забезпечити просту навігацію між основними екранами системи та швидкий доступ до функціональних можливостей застосунку. Під час розробки структури інтерфейсу враховувались особливості використання мобільних пристроїв та необхідність відображення великої кількості інформації у компактному вигляді [34].

Основними екранами мобільного застосунку є:

– екран авторизації;

– головний екран системи;

– екран інформації про автомобіль;

– екран телеметричних параметрів;

– екран статистики;

– екран налаштувань користувача.

Після запуску застосунку користувач переходить до екрану авторизації, де виконується вхід до системи. Після успішної авторизації користувач отримує доступ до основного функціоналу застосунку та може переглядати інформацію про власний транспортний засіб [23].

На рисунку 2.6 наведено загальну структуру мобільного застосунку.

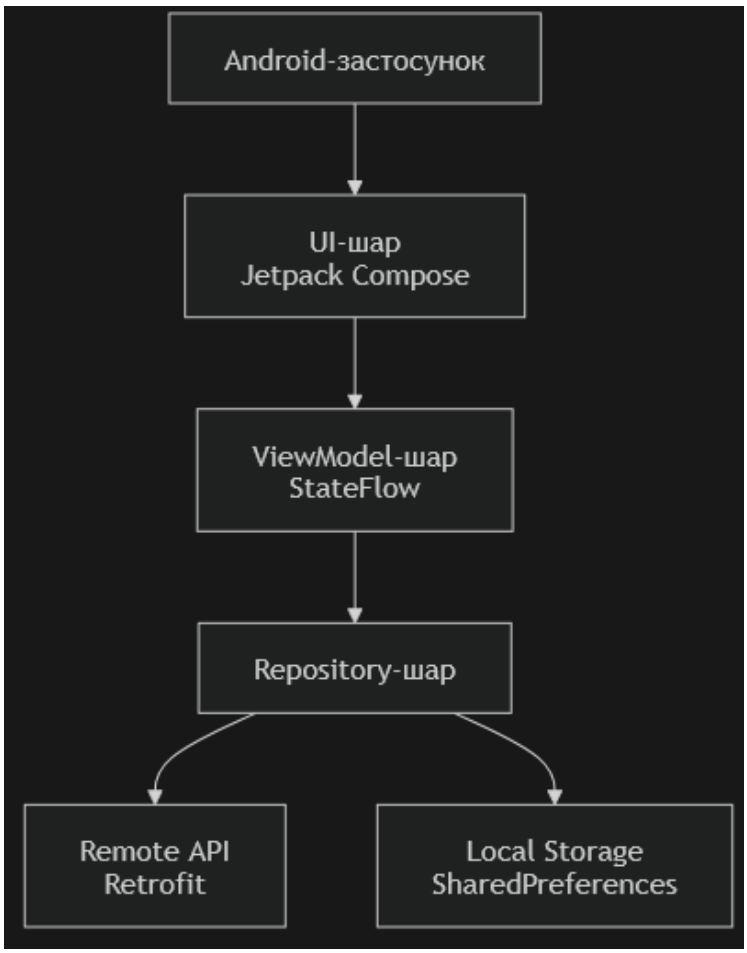

Рисунок 2.6 - Структура мобільного застосунку

Головний екран системи містить основну інформацію про автомобіль та дозволяє швидко перейти до перегляду телеметричних параметрів. На даному екрані можуть відображатись поточні показники автомобіля, стан підключення до системи та повідомлення про критичні зміни параметрів транспортного засобу.

Екран телеметричних параметрів використовується для перегляду інформації про роботу автомобіля у режимі реального часу. На даному екрані користувач може переглядати швидкість руху, температуру двигуна, рівень пального, оберти двигуна та інші параметри транспортного засобу [32].

Для зручності використання системи також передбачено окремий екран статистики, який дозволяє переглядати історію отриманих показників та аналізувати зміни параметрів автомобіля протягом певного періоду часу.

Навігація між основними екранами застосунку реалізована таким чином, щоб забезпечити швидкий доступ до необхідної інформації та спростити процес

взаємодії користувача із системою. Під час проєктування інтерфейсу також було враховано принципи Material Design, які використовуються для створення сучасних мобільних застосунків [35].

На рисунку 2.7 наведено схему навігації між основними екранами мобільного застосунку.

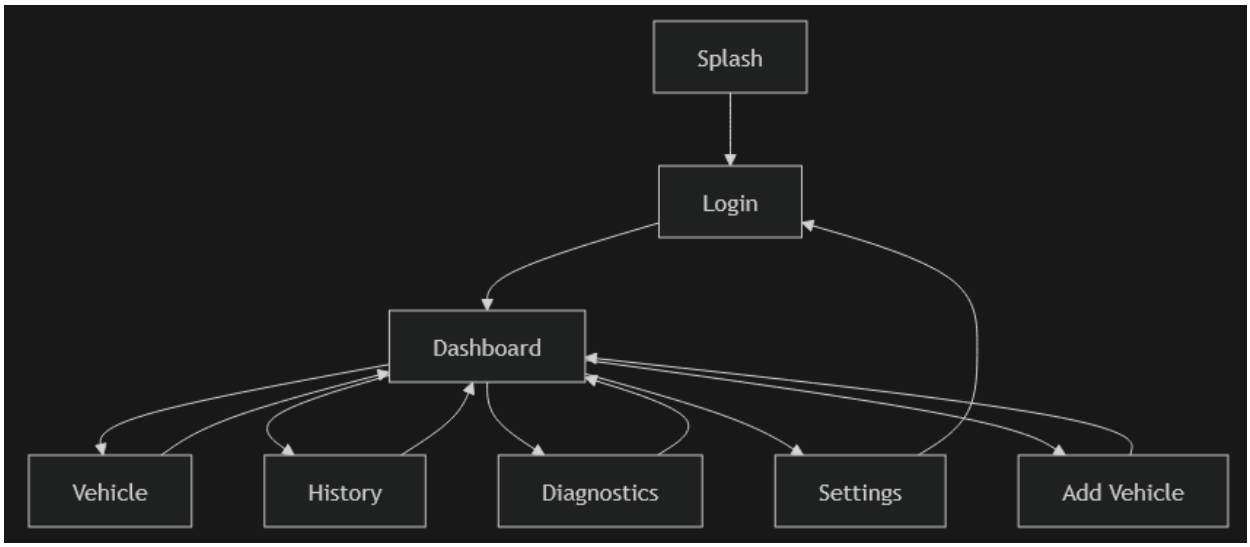

Рисунок 2.7 - Схема навігації мобільного застосунку

Під час проєктування структури мобільного застосунку також було враховано можливість подальшого розширення функціоналу системи. Це дозволяє у майбутньому додавати нові екрани, параметри моніторингу та додаткові функції без необхідності повної зміни структури застосунку.

2.7 Проєктування API та взаємодії компонентів системи

Для забезпечення взаємодії між мобільним застосунком, серверною частиною та базою даних у системі використовується API. Використання API дозволяє організувати передачу інформації між окремими компонентами системи та забезпечує стандартизований обмін даними [21].

Основним призначенням API є обробка запитів від клієнтської частини системи, передача інформації про користувачів та транспортні засоби, а також робота із телеметричними даними автомобіля. Передача інформації між компонентами системи здійснюється у форматі JSON, який широко використовується у сучасних клієнт-серверних системах [36].

У процесі роботи мобільний застосунок надсилає HTTP-запити до серверної частини системи. Сервер виконує обробку отриманого запиту, взаємодіє із базою даних та повертає результат до клієнтської частини застосунку [18].

На рисунку 2.8 наведено загальну схему взаємодії компонентів системи.

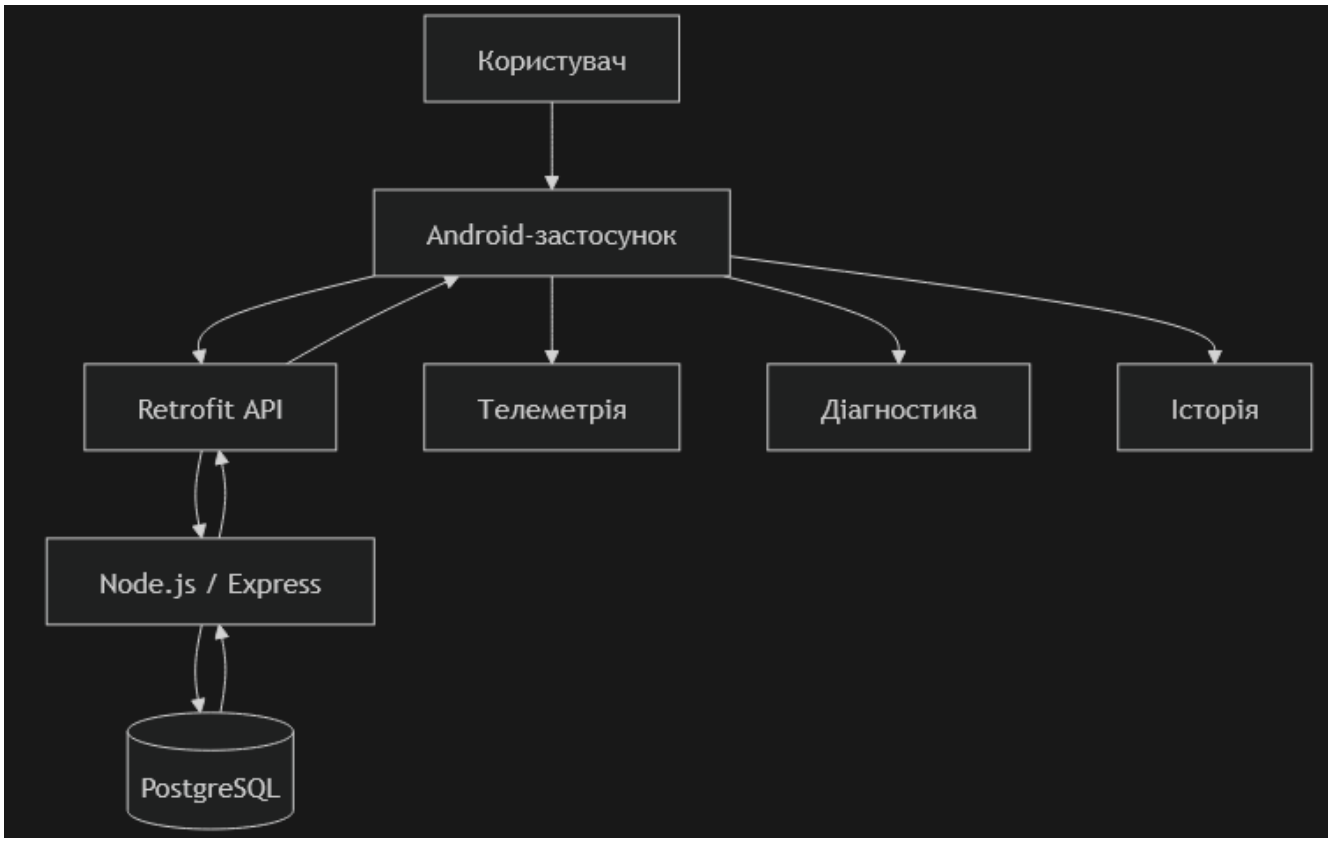

Рисунок 2.8 - Схема взаємодії компонентів системи

Основні API-запити, які використовуються у системі, наведено у таблиці 2.6.

Таблиця 2.6 - Основні API-запити системи

Метод	Призначення
POST	Авторизація користувача
GET	Отримання інформації про автомобіль
GET	Отримання телеметричних параметрів
POST	Передача телеметричних даних
GET	Отримання історії показників

Одним із основних API-запитів є авторизація користувача. Після введення користувачем даних мобільний застосунок надсилає POST-запит до серверної частини системи, після чого сервер виконує перевірку отриманої інформації та повертає результат авторизації [23].

Для отримання інформації про автомобіль та телеметричні параметри використовуються GET-запити. Серверна частина обробляє отримані запити, виконує звернення до бази даних та передає необхідну інформацію до мобільного застосунку [37].

Під час проєктування API також було враховано необхідність забезпечення стабільної передачі інформації між компонентами системи. Використання стандартизованих HTTP-запитів та формату JSON дозволяє спростити процес взаємодії між серверною та клієнтською частинами системи [36].

На рисунку 2.9 наведено схему обробки API-запитів у системі.

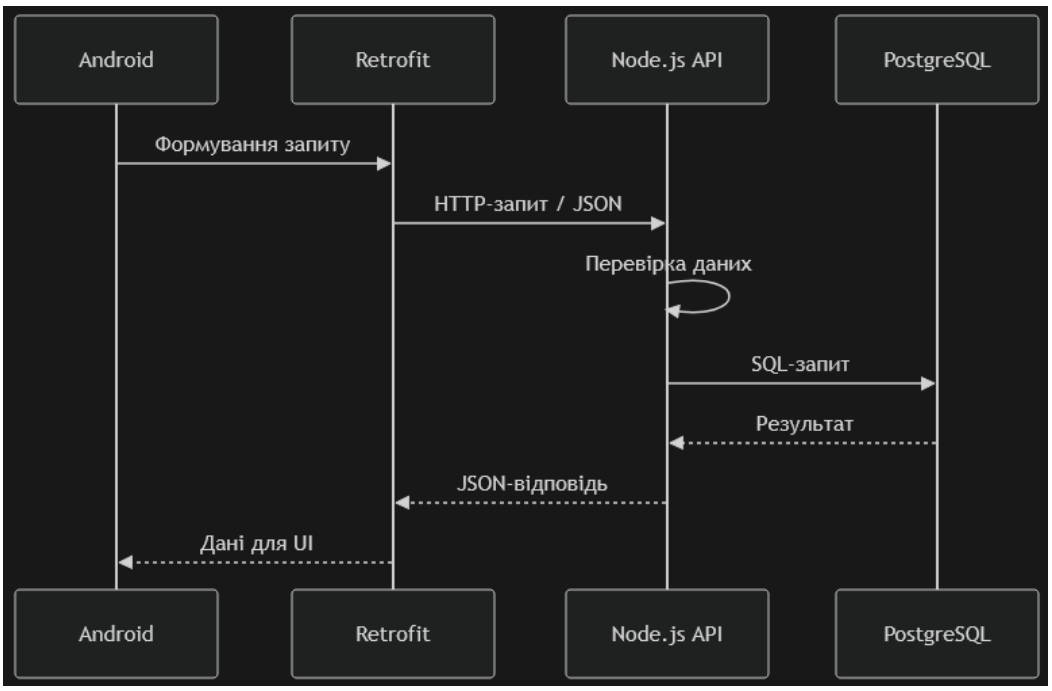

Рисунок 2.9 - Схема обробки API-запитів системи

2.8 Висновки до розділу

Проведене проєктування показало, що використання клієнт-серверної архітектури дозволяє ефективно розподілити функції між окремими компонентами системи. Мобільний застосунок забезпечує взаємодію користувача із системою та відображення телеметричних даних, серверна частина виконує обробку запитів і взаємодію з базою даних, а база даних забезпечує централізоване збереження інформації про користувачів, транспортні засоби та отримані показники. Такий підхід спрощує супровід програмного забезпечення та створює можливість його подальшого розвитку.

Особливу увагу під час проєктування було приділено процесу отримання та передачі телеметричної інформації. Було визначено загальну схему руху даних від автомобіля через інтерфейс OBD-II до мобільного застосунку та серверної частини системи. Це дозволило сформувати єдиний підхід до обробки

телеметричних параметрів і визначити вимоги до програмних компонентів, які беруть участь у цьому процесі.

Для опису функціональних можливостей системи було побудовано діаграму варіантів використання та визначено основні сценарії взаємодії користувача із програмним забезпеченням. Крім того, було розроблено ER-модель бази даних, яка дозволила визначити структуру збереження інформації та взаємозв'язки між основними сутностями системи. Отримані результати стали основою для побудови структури бази даних та реалізації серверної частини програмного забезпечення.

Таким чином, у результаті виконання другого розділу було сформовано повну модель комп'ютерної системи моніторингу стану автомобіля, визначено її архітектуру, структуру даних та механізми взаємодії між компонентами. Прийняті проєктні рішення стали основою для подальшої реалізації програмного забезпечення та інтеграції його окремих модулів у єдину систему.

3 РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ

3.1 Реалізація клієнтської частини системи

Реалізація клієнтської частини комп'ютерної системи моніторингу стану автомобіля виконувалась у середовищі Android Studio з використанням сучасних засобів розробки мобільних застосунків для операційної системи Android. Основною задачею клієнтської частини є забезпечення взаємодії користувача із системою, отримання телеметричних даних та відображення інформації про стан транспортного засобу [11].

Під час реалізації мобільного застосунку особлива увага приділялась структурі Android-проєкту, організації навігації між екранами, взаємодії із серверною частиною системи та створенню зручного користувацького інтерфейсу.

3.1.1 Структура Android-проєкту

Розробка мобільного застосунку виконувалась у середовищі Android Studio з використанням мови програмування Kotlin. Дане середовище розробки забезпечує підтримку сучасних інструментів створення Android-застосунків та дозволяє організувати структуру проєкту відповідно до вимог мобільної розробки [11]. Використання Android Studio також спрощує процес налагодження програмного забезпечення, тестування окремих модулів та контролю коректності роботи застосунку на різних версіях операційної системи Android.

Під час реалізації клієнтської частини системи структура Android-проєкту була розділена на окремі компоненти, що відповідають за інтерфейс користувача, роботу із даними, взаємодію із серверною частиною та обробку телеметричних параметрів автомобіля. Такий підхід дозволяє розподілити функціональність між окремими програмними модулями, підвищити зручність супроводу програмного коду та спростити подальше розширення функціональних можливостей системи.

Основними компонентами структури проєкту є:

– модулі екранів застосунку;

– класи ViewModel;

– модулі роботи із API;

– моделі даних;

– навігаційні компоненти;

– компоненти користувацького інтерфейсу.

Поділ проєкту на окремі логічні компоненти дозволяє забезпечити незалежність між різними частинами застосунку та спрощує взаємодію між ними. Крім того, така структура відповідає сучасним підходам до розробки Android-застосунків та сприяє повторному використанню програмного коду в межах проєкту.

На рисунку 3.1 наведено загальну структуру Android-проєкту.

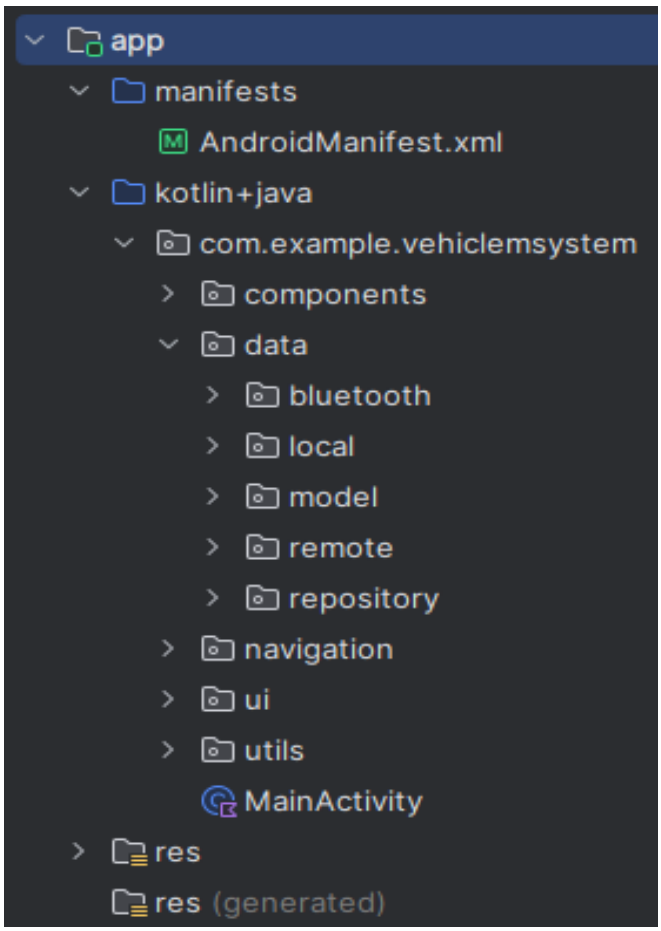

Рисунок 3.1 - Структура Android-проєкту

Для зручності підтримки програмного коду окремі компоненти застосунку були розподілені по відповідних пакетах. Подібний підхід дозволяє спростити процес розробки, покращити читабельність структури проєкту та забезпечити зручну взаємодію між окремими модулями системи [39].

Пакет ui містить екрани мобільного застосунку та компоненти користувацького інтерфейсу. У даному пакеті реалізовано основні екрани системи, включаючи екран авторизації, головний екран, екран телеметричних параметрів та екран статистики.

Пакет viewmodel використовується для реалізації логіки взаємодії між інтерфейсом користувача та даними системи. У даному пакеті розміщуються класи ViewModel, які забезпечують обробку отриманої інформації та збереження стану інтерфейсу [39].

Пакет network містить компоненти, що відповідають за взаємодію із серверною частиною системи. У межах даного пакету реалізовано API-запити, обробку HTTP-відповідей та механізми передачі інформації між клієнтською та серверною частинами системи [40].

Для збереження структури телеметричних даних та інформації про транспортний засіб використовуються окремі моделі даних. Використання моделей дозволяє спростити процес обробки інформації та забезпечити зручну взаємодію між компонентами системи.

Приклад моделі даних наведено у лістингу 3.1.

Лістинг 3.1 - Приклад моделі телеметричних даних

```
data class TelemetryData(
    val speed: Int,
    val rpm: Int,
    val engineTemp: Int,
    val batteryVoltage: Float,
    val timestamp: String
)
```

У лістингу 3.1 представлено модель телеметричних даних, яка використовується для збереження та передачі параметрів стану автомобіля у мобільному застосунку. Модель містить основні показники телеметрії, зокрема швидкість руху, оберти двигуна, температуру двигуна, напругу акумулятора та часову позначку отримання даних.

3.1.2 Реалізація навігації між екранами

Для забезпечення зручної взаємодії користувача із мобільним застосунком у системі було реалізовано навігацію між основними екранами застосунку. Навігація дозволяє організувати логічний перехід між окремими компонентами системи та забезпечує швидкий доступ до основного функціоналу мобільного застосунку [38].

Під час реалізації навігації використовувався Navigation Component, який є одним із стандартних засобів навігації у середовищі Android Studio. Використання

даного компонента дозволяє централізовано керувати переходами між екранами та спрощує процес підтримки структури застосунку [38].

Основними екранами мобільного застосунку є:

– екран авторизації;

– головний екран;

– екран перегляду телеметричних параметрів;

– екран статистики;

– екран налаштувань користувача.

Після запуску застосунку користувач переходить до екрана авторизації. Після успішного входу до системи виконується перехід до головного екрана, який містить основну інформацію про стан автомобіля та забезпечує доступ до інших функцій системи [23].

Головний екран використовується для перегляду поточних параметрів автомобіля та переходу до додаткових екранів системи. Із головного екрана користувач може перейти до перегляду статистики, історії показників або налаштувань застосунку.

Для реалізації навігації використовувався NavHost, який забезпечує керування маршрутами між окремими екранами застосунку.

Приклад реалізації NavHost наведено у лістингу 3.2.

Лістинг 3.2 - Реалізація навігації між екранами

```
@Composable
fun NavGraph() {
    val navController =
        rememberNavController()
    NavHost(
        navController = navController,
        startDestination =
            Routes.Splash.route
    ) {
        composable(
            Routes.Splash.route
        ) {
            SplashScreen(navController)
        }
        composable(
            Routes.Login.route
```

```
        ) {
            LoginScreen(navController)
        }
        composable(
            Routes.Dashboard.route
        ) {
            DashboardScreen(navController)
        }
        composable(
            Routes.Vehicle.route
        ) {
            VehicleScreen()
        }
        composable(
            Routes.Diagnostics.route
        ) {
            DiagnosticsScreen(navController)
        }
        composable(
            Routes.History.route
        ) {
            HistoryScreen(navController)
        }
        composable(
            Routes.Settings.route
        ) {
            SettingsScreen(navController)
        }
    }
}
```

У наведеному лістингу реалізовано навігацію між основними екранами мобільного застосунку. Початковим екраном системи є екран авторизації, після якого користувач отримує доступ до інших компонентів застосунку.

Для переходу між екранами використовується об’єкт navController, який забезпечує керування маршрутами та дозволяє виконувати переходи між окремими компонентами системи.

Під час реалізації навігації також було враховано необхідність забезпечення зручної взаємодії користувача із застосунком. Основні елементи навігації розташовані таким чином, щоб користувач мав швидкий доступ до основного функціоналу системи та міг легко переміщуватись між екранами застосунку.

На рисунку 3.2 наведено приклад переходу між екранами мобільного застосунку.

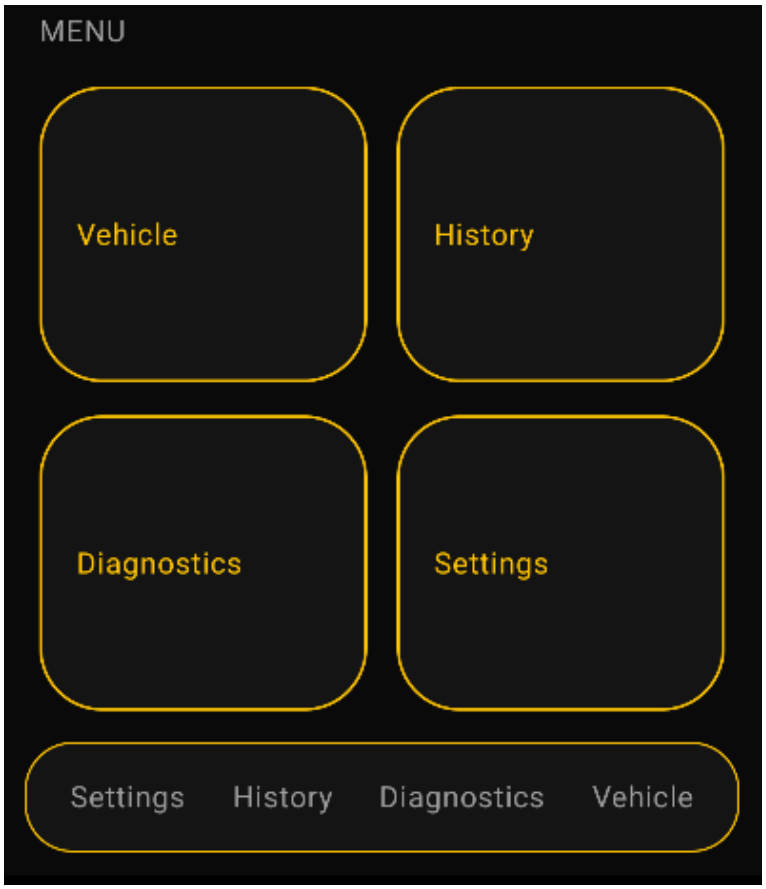

Рисунок 3.2 - Приклад навігації у мобільному застосунку

Використання Navigation Component дозволило реалізувати гнучку структуру навігації та спростити процес подальшого розширення функціоналу мобільного застосунку.

3.1.3 Реалізація ViewModel та логіки застосунку

Для реалізації логіки мобільного застосунку використовувався архітектурний підхід MVVM (Model–View–ViewModel), який дозволяє розділити компоненти інтерфейсу користувача, логіку обробки даних та механізми взаємодії із серверною частиною системи [39].

Використання архітектури MVVM дозволяє спростити підтримку програмного коду, забезпечити повторне використання окремих компонентів та зменшити залежність між інтерфейсом користувача і логікою застосунку.

Основними компонентами архітектури MVVM є:

– Model;

– View;

– ViewModel.

Компонент Model використовується для збереження та обробки даних системи. У межах мобільного застосунку модель містить інформацію про транспортний засіб, телеметричні параметри та дані користувача.

Компонент View відповідає за відображення інформації користувачу. До даного компонента належать екрани мобільного застосунку та елементи користувацького інтерфейсу.

ViewModel використовується для взаємодії між інтерфейсом користувача та моделями даних. Основною задачею ViewModel є обробка отриманої інформації та збереження стану інтерфейсу під час роботи застосунку [39].

На рисунку 3.3 наведено схему взаємодії компонентів MVVM у мобільному застосунку.

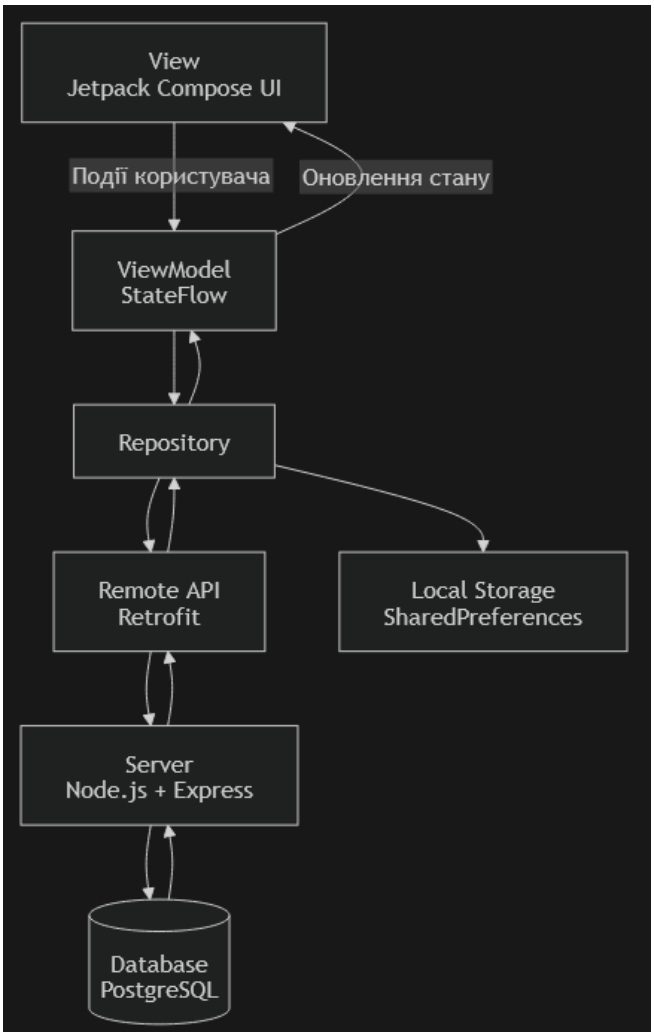

Рисунок 3.3 - Схема взаємодії компонентів MVVM

Під час реалізації клієнтської частини системи для кожного основного екрана було створено окремий ViewModel-клас. Подібний підхід дозволяє розділити логіку роботи окремих компонентів застосунку та спростити процес обробки інформації.

ViewModel використовується для:

– отримання телеметричних даних;

– обробки API-відповідей;

– оновлення стану інтерфейсу;

– передачі інформації до UI-компонентів;

– збереження поточного стану застосунку.

Однією з основних задач ViewModel є отримання телеметричних параметрів автомобіля та їх подальша передача до компонентів користувацького інтерфейсу. Для реалізації даного механізму використовується окремий клас ObdReader, який забезпечує взаємодію мобільного застосунку із адаптером ELM327 через Bluetooth-з’єднання та виконує отримання даних від електронного блока керування автомобіля [29].

Отримані параметри надалі передаються до ViewModel, де виконується оновлення стану застосунку та відображення інформації на екранах мобільного застосунку. Подібний підхід дозволяє розділити логіку отримання телеметричних даних та логіку керування станом інтерфейсу відповідно до принципів архітектури MVVM [39].

Приклад реалізації класу для отримання телеметричних параметрів через адаптер ELM327 наведено у лістингу 3.3.

Лістинг 3.3 - Реалізація отримання телеметричних даних через адаптер ELM327

```
class ObdReader(
    private val socket: BluetoothSocket
) {

    private val input =
        socket.inputStream
```

```
private val output =
    socket.outputStream

private fun sendCommand(command: String): String {

    output.write(
        "$command\r".toByteArray()
    )

    output.flush()

    val buffer =
        ByteArray(1024)

    val count =
        input.read(buffer)

    return String(
        buffer,
        0,
        count
    )
}

fun readSpeed(): Int {

    val response =
        sendCommand("010D")

    val bytes =
        response
            .replace(" ", "")
            .replace(">", "")

    val value =
        bytes.takeLast(2)

    return value.toInt(16)
}

fun readRpm(): Int {

    val response =
        sendCommand("010C")

    val bytes =
        response
            .replace(" ", "")
            .replace(">", "")

    val a =
        bytes.substring(
            bytes.length - 4,
```

```
                bytes.length - 2
            ).toInt(16)

        val b =
            bytes.takeLast(2)
                .toInt(16)

        return ((a * 256) + b) / 4
    }

    fun readEngineTemp(): Int {

        val response =
            sendCommand("0105")

        val bytes =
            response
                .replace(" ", "")
                .replace(">", "")

        return bytes.takeLast(2)
            .toInt(16) - 40
    }
}
```

У наведеному лістингу реалізовано клас ObdReader, який використовується для отримання телеметричних параметрів автомобіля через адаптер ELM327. Для обміну інформацією використовується Bluetooth-з'єднання, що забезпечує передачу команд від мобільного застосунку до адаптера та отримання відповідей від електронного блока керування автомобіля (ECU) [29].

Основною задачею класу є формування OBD-II запитів та обробка відповідей, отриманих від транспортного засобу. Для взаємодії з адаптером використовуються вхідний та вихідний потоки BluetoothSocket. Передача команд виконується за допомогою методу sendCommand, який надсилає PID-запит до адаптера ELM327 та отримує відповідь у текстовому форматі [6].

Для отримання окремих параметрів автомобіля використовуються стандартні PID-команди протоколу OBD-II. Команда 010D використовується для отримання швидкості руху транспортного засобу, команда 010C дозволяє отримати інформацію про оберти двигуна, а команда 0105 використовується для визначення температури охолоджувальної рідини двигуна [7].

Після отримання відповіді від адаптера виконується її попередня обробка. Із рядка видаляються службові символи та пробіли, після чого отримані шістнадцяткові значення перетворюються у числовий формат відповідно до специфікації стандарту OBD-II. Отримані значення можуть бути використані для подальшого відображення у користувацькому інтерфейсі мобільного застосунку та передачі до серверної частини системи.

Для роботи користувацького інтерфейсу отримані телеметричні параметри передаються до відповідних ViewModel-класів, де виконується оновлення поточного стану застосунку за допомогою механізму StateFlow. Використання такого підходу дозволяє автоматично оновлювати інформацію на екранах мобільного застосунку після надходження нових даних від автомобіля [35].

Застосування адаптера ELM327 у поєднанні зі стандартом OBD-II дозволяє отримувати інформацію про стан транспортного засобу без внесення змін у його конструкцію та забезпечує сумісність із більшістю сучасних автомобілів. Отримані телеметричні параметри можуть використовуватись для моніторингу роботи автомобіля, формування статистики та своєчасного виявлення можливих відхилень у роботі основних систем транспортного засобу [29].

Використання архітектури MVVM у поєднанні із механізмами асинхронної обробки даних дозволило організувати ефективну взаємодію між компонентами застосунку, забезпечити автоматичне оновлення користувацького інтерфейсу та спростити подальше розширення функціональних можливостей системи [39].

3.1.4 Реалізація API-запитів

Для забезпечення взаємодії між мобільним застосунком та серверною частиною системи використовувались API-запити. API забезпечує передачу інформації між окремими компонентами системи та дозволяє отримувати телеметричні дані автомобіля, інформацію про користувача та інші параметри системи [21].

Передача інформації між клієнтською та серверною частинами здійснюється за допомогою HTTP-запитів у форматі JSON. Використання

формату JSON дозволяє спростити обробку інформації та забезпечити сумісність між різними компонентами системи [36].

Для реалізації API-запитів у мобільному застосунку використовувалась бібліотека Retrofit, яка забезпечує зручний механізм взаємодії із серверною частиною системи та підтримує автоматичну обробку HTTP-відповідей [40].

На рисунку 3.4 наведено загальну схему взаємодії мобільного застосунку із API.

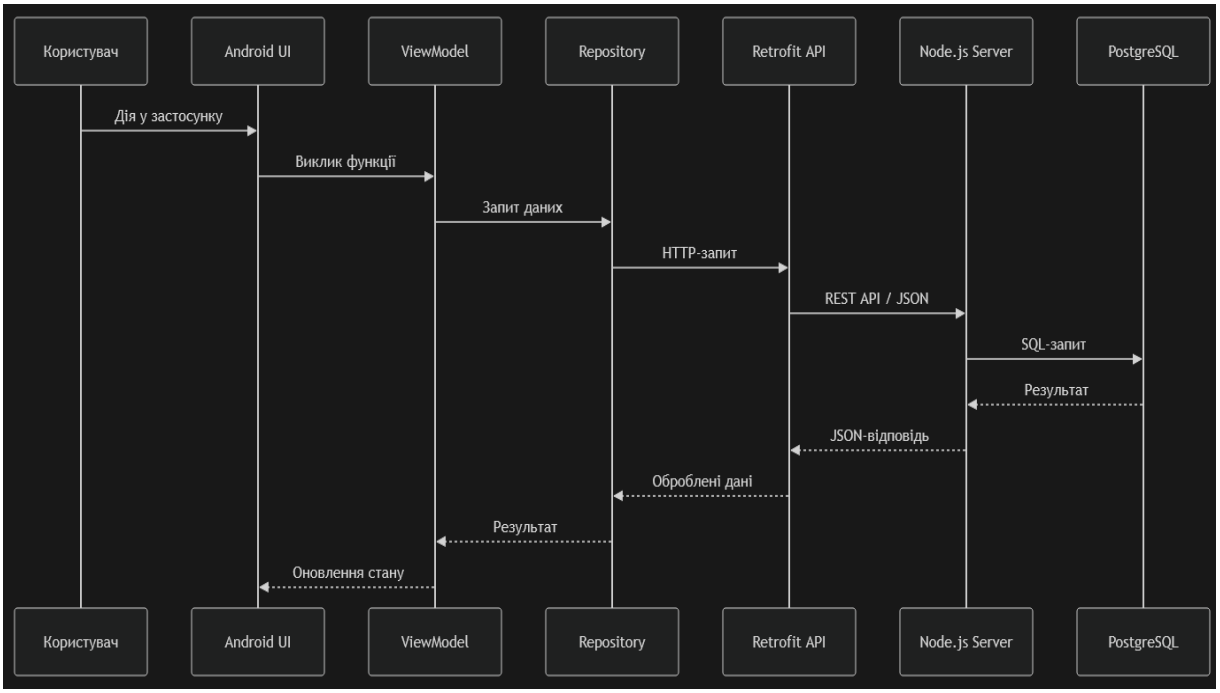

Рисунок 3.4 - Схема взаємодії мобільного застосунку із API

Основними API-запитами, які використовуються у системі, є:

– авторизація користувача;

– отримання інформації про автомобіль;

– отримання телеметричних параметрів;

– передача телеметричних даних;

– отримання статистичної інформації.

Для реалізації API-запитів було створено окремий інтерфейс, який містить методи взаємодії із серверною частиною системи.

Приклад реалізації API-інтерфейсу наведено у лістингу 3.4.

Лістинг 3.4 - Реалізація API-інтерфейсу

```
interface AuthApi {
    @POST("api/auth/login")
    suspend fun login(

        @Body request: LoginRequest
    ): Response<LoginResponse>
    @POST("api/auth/register")
    suspend fun register(
        @Body request: LoginRequest
    ): Response<LoginResponse>
}
```

У наведеному лістингу представлено реалізацію API-інтерфейсу для взаємодії мобільного застосунку із серверною частиною системи. Інтерфейс містить методи авторизації та реєстрації користувача за допомогою HTTP-запитів типу POST. Для передачі даних використовується об’єкт LoginRequest, а відповідь сервера обробляється у вигляді LoginResponse.

Для створення HTTP-клієнта та підключення до серверної частини використовувався Retrofit.Builder.

Приклад створення Retrofit-клієнта наведено у лістингу 3.5.

Лістинг 3.5 - Створення Retrofit-клієнта

```
object RetrofitClient {
    private const val BASE_URL =
        "http://10.0.2.2:3000/"
    private val retrofit =
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(
                GsonConverterFactory.create()
            )
            .build()
    val authApi: AuthApi =
        retrofit.create(AuthApi::class.java)
}
```

У наведеному лістингу задається базова адреса серверної частини системи Метод використовується для організації мережевої взаємодії між мобільним застосунком та сервером. Клас RetrofitClient забезпечує створення HTTP-клієнта

із заданою базовою адресою сервера та автоматичним перетворенням JSON-відповідей у Kotlin-об’єкти.

3.1.5 Реалізація користувацького інтерфейсу

Під час реалізації клієнтської частини системи значна увага приділялась створенню зручного та сучасного користувацького інтерфейсу. Основною задачею інтерфейсу є забезпечення швидкого доступу до інформації про стан автомобіля та спрощення взаємодії користувача із функціональними можливостями системи [35].

Для побудови інтерфейсу мобільного застосунку використовувався Jetpack Compose, який дозволяє створювати UI-компоненти за допомогою функцій Kotlin та забезпечує сучасний підхід до розробки Android-застосунків [35].

Використання Jetpack Compose дозволило:

– спростити процес створення інтерфейсу;

– реалізувати автоматичне оновлення компонентів;

– забезпечити підтримку сучасного дизайну застосунку.

Основними елементами користувацького інтерфейсу є:

– інформаційні картки;

– кнопки навігації;

– поля введення даних;

– панелі параметрів автомобіля;

– елементи статистики та графіків.

На рисунку 3.5 наведено приклад структури користувацького інтерфейсу мобільного застосунку.

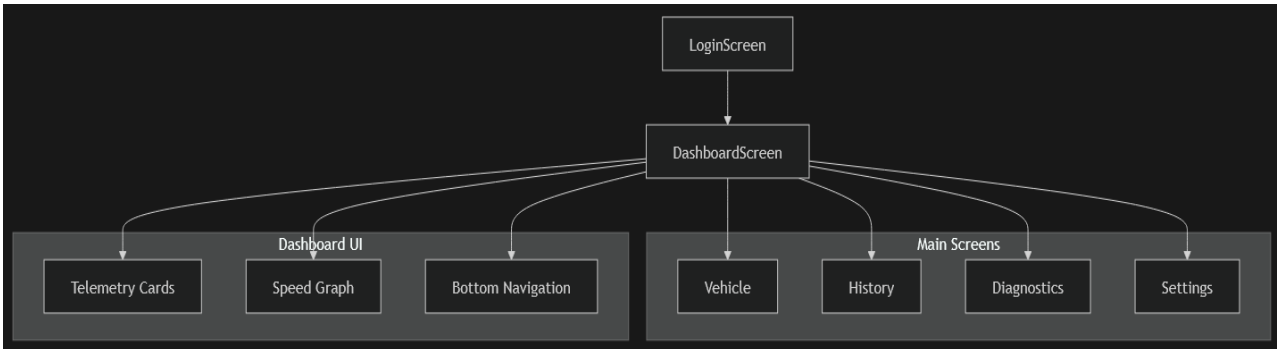

Рисунок 3.5 - Структура користувацького інтерфейсу застосунку

Для відображення телеметричних параметрів автомобіля використовувались окремі інформаційні блоки, які містять основні показники транспортного засобу. Подібний підхід дозволяє спростити сприйняття інформації користувачем та забезпечує швидкий доступ до важливих параметрів автомобіля.

Приклад реалізації інформаційної картки наведено у лістингу 3.6.

Лістинг 3.6 - Реалізація інформаційної картки параметра автомобіля

```
@Composable
fun TelemetryCard(
    title: String,
    value: String
) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .border(
                1.dp,
                AccentYellow,
                RoundedCornerShape(20.dp)
            ),
        shape = RoundedCornerShape(20.dp)
    ) {
        Column(
            modifier = Modifier
                .background(CardDark)
                .padding(18.dp)
        ) {
            Text(
                text = title,
                color = TextSecondary
            )
            Text(
                text = value,
                color = AccentYellow
            )
        }
    }
}
```

У наведеному лістингу реалізовано інформаційну картку для відображення телеметричних параметрів автомобіля. Для побудови компонента використовується елемент Card, який дозволяє групувати інформацію у вигляді окремих блоків інтерфейсу.

Для розміщення елементів усередині картки використовується компонент Column, що забезпечує вертикальне розташування текстових елементів. Значення телеметричних параметрів автоматично оновлюються після зміни стану системи та відображаються у реальному часі.

Під час реалізації користувацького інтерфейсу використовувались принципи Material Design, які забезпечують єдиний стиль компонентів та покращують зручність взаємодії користувача із системою [35].

Для забезпечення адаптивності інтерфейсу використовувались стандартні компоненти Jetpack Compose та механізми автоматичного оновлення UI-компонентів. Це дозволяє коректно відображати інформацію на мобільних пристроях із різними розмірами екранів.

Основні параметри автомобіля відображаються у вигляді окремих інформаційних блоків, що дозволяє користувачу швидко отримувати інформацію про поточний стан транспортного засобу.

Використання Jetpack Compose та компонентів Material Design дозволило реалізувати сучасний користувацький інтерфейс та забезпечити зручну взаємодію користувача із системою моніторингу стану автомобіля.

3.2 Реалізація серверної частини системи

Серверна частина комп’ютерної системи моніторингу стану автомобіля використовується для обробки запитів від мобільного застосунку, взаємодії із базою даних та передачі телеметричної інформації між окремими компонентами системи. Основною задачею серверної частини є забезпечення стабільного обміну даними між клієнтською частиною та базою даних [18].

Під час реалізації серверної частини системи було враховано необхідність підтримки авторизації користувачів, отримання телеметричних параметрів автомобіля та збереження історії показників транспортного засобу.

Взаємодія між мобільним застосунком та сервером здійснюється за допомогою API-запитів. Передача інформації виконується у форматі JSON, який забезпечує зручну структуру обміну даними та підтримується більшістю сучасних інформаційних систем [36].

На рисунку 3.6 наведено загальну структуру серверної частини системи.

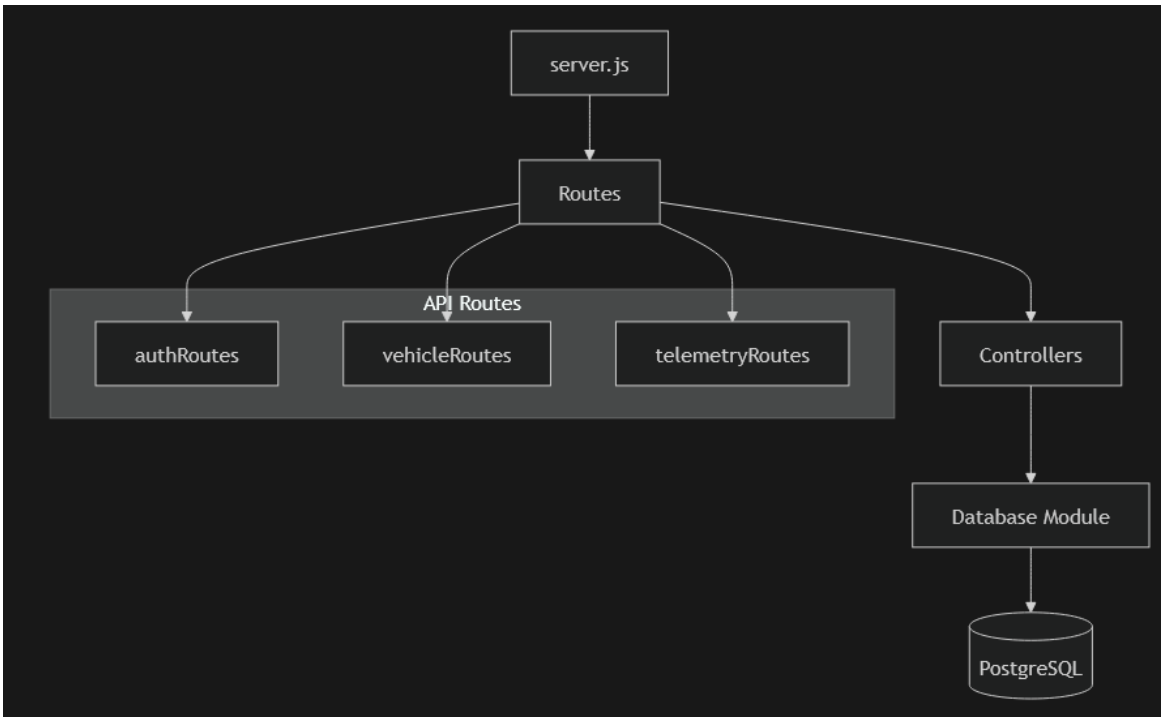

Рисунок 3.6 - Структура серверної частини системи

Серверна частина системи виконує наступні функції:

– обробка HTTP-запитів;

– авторизація користувачів;

– отримання телеметричних параметрів;

– взаємодія із базою даних;

– передача інформації до мобільного застосунку;

– збереження історії телеметричних даних.

Для реалізації API використовувались окремі маршрути, які забезпечують обробку запитів від клієнтської частини системи.

Приклад реалізації маршруту авторизації наведено у лістингу 3.7.

Лістинг 3.7 - Реалізація маршруту авторизації користувача

```
router.post(
    "/login",
    async (req, res) => {
        const { email, password } = req.body
        const result = await pool.query(

            `
            SELECT * FROM users
            WHERE email = $1
            `,

            [email]
        )
        if(result.rows.length === 0) {
            return res.status(401).json({
                error: "Invalid email"
            })
        }
        const user = result.rows[0]
        const validPassword =
            await bcrypt.compare(
                password,
                user.password
            )
        if(!validPassword) {
            return res.status(401).json({
                error: "Invalid password"
            })
        }
        res.json({
            token: "jwt_token",
            user: {
                id: user.id,
                email: user.email
            }
        })
    }
)
```

У наведеному лістингу реалізовано POST-запит авторизації користувача. Серверна частина отримує дані користувача, виконує перевірку електронної пошти та пароля, після чого повертає результат авторизації до мобільного застосунку.

Для перевірки пароля використовується бібліотека bcryptjs, яка дозволяє виконувати порівняння зашифрованих паролів та підвищує безпеку роботи системи.

Для передачі телеметричних параметрів автомобіля використовуються окремі API-запити, які дозволяють отримувати інформацію про стан транспортного засобу та передавати її до клієнтської частини системи.

Приклад реалізації GET-запиту для отримання телеметричних параметрів наведено у лістингу 3.8.

Лістинг 3.8 - Реалізація GET-запиту для отримання телеметричних даних

```
router.get(
    "/:user_id",
    async (req, res) => {
        const { user_id } = req.params
        const result = await pool.query(
            `
            SELECT * FROM vehicles
            WHERE user_id = $1
            LIMIT 1
            `,
            [user_id]
        )
        res.json(result.rows[0])
    }
)
```

У наведеному лістингу реалізовано GET-запит для отримання інформації про автомобіль користувача. Сервер формує відповідь у форматі JSON та передає клієнтській частині дані про транспортний засіб, зокрема марку, модель та VIN-код автомобіля.

Для взаємодії із базою даних серверна частина використовує окремі модулі обробки даних. Після отримання запиту сервер виконує звернення до бази даних, отримує необхідну інформацію та передає результат до клієнтської частини застосунку [30].

Під час реалізації серверної частини також було враховано необхідність обробки помилок під час роботи системи. У випадку виникнення помилки сервер повертає відповідне повідомлення про помилку або інформацію про неможливість виконання запиту.

Для забезпечення стабільної роботи системи всі запити проходять перевірку коректності отриманих даних. Це дозволяє уникнути передачі некоректної інформації та підвищує надійність роботи системи.

Використання серверної частини дозволило реалізувати централізовану обробку інформації, забезпечити взаємодію між компонентами системи та організувати збереження телеметричних параметрів автомобіля.

3.3 Реалізація бази даних системи

База даних у комп'ютерній системі моніторингу стану автомобіля використовується для централізованого збереження інформації про користувачів, транспортні засоби та телеметричні параметри автомобіля. Використання бази даних дозволяє забезпечити структуроване збереження інформації, спростити процес обробки даних та реалізувати механізм накопичення історії телеметричних показників [30].

Під час реалізації системи структура бази даних була побудована відповідно до ER-моделі згідно з рисунком 2.4, розробленої на етапі проєктування системи. Основними таблицями бази даних є:

– таблиця користувачів;

– таблиця автомобілів;

– таблиця телеметричних даних.

Таблиця користувачів використовується для збереження інформації про зареєстрованих користувачів системи. У даній таблиці зберігаються логін користувача, адреса електронної пошти та пароль.

Приклад створення таблиці користувачів наведено у лістингу 3.9.

Лістинг 3.9 - Створення таблиці користувачів

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
```

```
    email VARCHAR(255)
        UNIQUE NOT NULL,
    password VARCHAR(255)
        NOT NULL,
    created_at TIMESTAMP
        DEFAULT CURRENT_TIMESTAMP
);
```

У наведеному лістингу створюється таблиця users, яка містить основну інформацію про користувачів системи. Поле id використовується як унікальний ідентифікатор користувача, а поле email забезпечує збереження електронної адреси користувача та контроль унікальності облікових записів.

Для збереження інформації про транспортні засоби використовується окрема таблиця автомобілів. У даній таблиці міститься інформація про марку автомобіля, модель, рік випуску та зв’язок із конкретним користувачем системи.

Приклад створення таблиці автомобілів наведено у лістингу 3.10.

Лістинг 3.10 - Створення таблиці автомобілів

```
CREATE TABLE vehicles (
    id SERIAL PRIMARY KEY,
    user_id INTEGER
        REFERENCES users(id),
    vin VARCHAR(50),
    brand VARCHAR(100),
    model VARCHAR(100),
    year INTEGER
);
```

У наведеному лістингу реалізовано таблицю vehicles, яка містить інформацію про транспортні засоби користувачів. Для встановлення зв’язку між автомобілем та користувачем використовується зовнішній ключ user_id.

Для збереження телеметричних параметрів автомобіля використовується окрема таблиця telemetry. У даній таблиці зберігаються параметри транспортного засобу, отримані під час роботи системи моніторингу.

Приклад створення таблиці телеметричних даних наведено у лістингу 3.11.

Лістинг 3.11 - Створення таблиці телеметричних даних

```
CREATE TABLE telemetry (
    id SERIAL PRIMARY KEY,
    vehicle_id INTEGER
        REFERENCES vehicles(id),
    speed INTEGER,
    rpm INTEGER,
    engine_temp INTEGER,
    battery_voltage REAL,
    created_at TIMESTAMP
        DEFAULT CURRENT_TIMESTAMP
);
```

У наведеному лістингу реалізовано таблицю telemetry, яка використовується для збереження інформації про стан транспортного засобу. У таблиці містяться поля для збереження швидкості руху, температури двигуна, обертів двигуна та напруги акумулятора.

Під час реалізації бази даних також було враховано необхідність збереження історії телеметричних параметрів автомобіля. Для цього кожний запис у таблиці telemetry містить дату та час отримання інформації.

Для взаємодії серверної частини із базою даних використовуються SQL-запити, які забезпечують додавання, отримання та оновлення інформації у системі [30].

Приклад SQL-запиту для отримання телеметричних параметрів автомобіля наведено у лістингу 3.12.

Лістинг 3.12 - SQL-запит для отримання телеметричних даних

```
SELECT *
FROM telemetry
WHERE vehicle_id = 1
ORDER BY created_at DESC;
```

У наведеному лістингу виконується отримання телеметричних параметрів для конкретного автомобіля із сортуванням записів за датою отримання інформації.

Лістинг 3.13 - Обробка SQL-запиту на сервері

```
const result = await pool.query(
    `
    SELECT * FROM telemetry
    WHERE vehicle_id = $1
    ORDER BY created_at DESC
    `,
    [vehicleId]
)
res.json(result.rows)
```

Під час реалізації бази даних також було враховано можливість подальшого розширення структури системи. Використання окремих таблиць для користувачів, автомобілів та телеметричних параметрів дозволяє спростити додавання нових функціональних можливостей та параметрів моніторингу без необхідності повної зміни структури бази даних.

3.4 Реалізація повноцінного інтерфейсу

Користувацький інтерфейс комп'ютерної системи моніторингу стану автомобіля реалізовано у вигляді мобільного застосунку для операційної системи Android. Основною задачею інтерфейсу є забезпечення зручної взаємодії користувача із системою та відображення телеметричних параметрів автомобіля у зрозумілому вигляді [35].

Під час реалізації інтерфейсу використовувався Jetpack Compose, який дозволяє створювати сучасні UI-компоненти за допомогою мови програмування Kotlin. Використання Jetpack Compose спрощує процес створення інтерфейсу та забезпечує автоматичне оновлення елементів після зміни даних [35].

Основними екранами мобільного застосунку є:

– екран авторизації;

– головний екран;

– екран телеметричних параметрів;

– екран статистики;

– екран налаштувань користувача.

3.4.1 Реалізація екрана авторизації

Екран авторизації використовується для входу користувача до системи та забезпечення доступу до функціональних можливостей мобільного застосунку. Для авторизації користувач вводить логін та пароль, після чого мобільний застосунок надсилає відповідний запит до серверної частини системи [23].

Для побудови екрана авторизації використовувались компоненти TextField та Button, які забезпечують введення інформації та взаємодію користувача із системою.

Приклад реалізації поля введення наведено у лістингу 3.14.

Лістинг 3.14 - Реалізація поля введення даних користувача

```
OutlinedTextField(
    value = state.email,
    onValueChange = {
        viewModel.updateEmail(it)
    },
    modifier = Modifier
        .fillMaxWidth(),
    label = {
        Text("Email")
    }
)
```

У наведеному лістингу реалізовано текстове поле для введення електронної адреси користувача. Після зміни значення виконується оновлення стану loginState, який використовується під час авторизації користувача у системі.

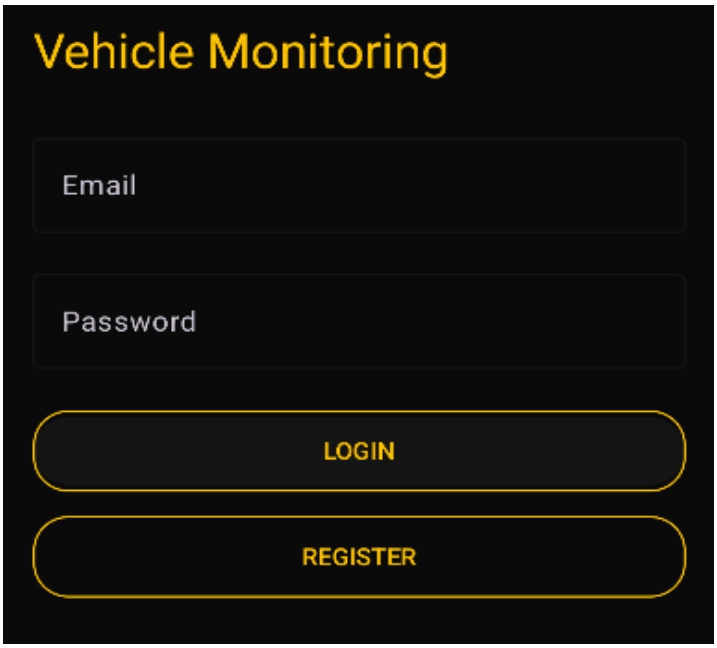

Рисунок 3.7 - Екран авторизації системи

3.4.2 Реалізація головного екрана

Головний екран використовується для відображення основної інформації про стан автомобіля та швидкого доступу до функціональних можливостей системи. На головному екрані відображаються основні телеметричні параметри транспортного засобу та інформація про поточний стан системи. Крім цього, користувач може швидко перейти до перегляду історії показників, діагностики автомобіля, налаштувань застосунку та керування інформацією про транспортний засіб.

Для побудови інтерфейсу використовувались компоненти Card та Column, які дозволяють відображати інформацію у вигляді окремих інформаційних блоків. Такий підхід забезпечує зручне групування телеметричних даних та покращує сприйняття інформації користувачем. Використання компонентів Jetpack Compose також дозволяє автоматично оновлювати інтерфейс після зміни стану даних без необхідності ручного оновлення окремих елементів екрана.

Приклад реалізації інформаційної картки наведено у лістингу 3.15.

Лістинг 3.15 - Реалізація інформаційної картки

```
TelemetryCard(
title = "Engine Temperature",
value = "${state.engineTemp}°C"
)
```

У наведеному лістингу реалізовано інформаційну картку для відображення температури двигуна автомобіля. Поточне значення параметра отримується із об’єкта state та автоматично оновлюється після зміни телеметричних даних. Використання окремого компонента для відображення параметрів дозволяє повторно використовувати його для виведення інших показників автомобіля, зокрема швидкості руху, обертів двигуна та напруги акумулятора.

На рисунку 3.8 наведено приклад головного екрана мобільного застосунку.

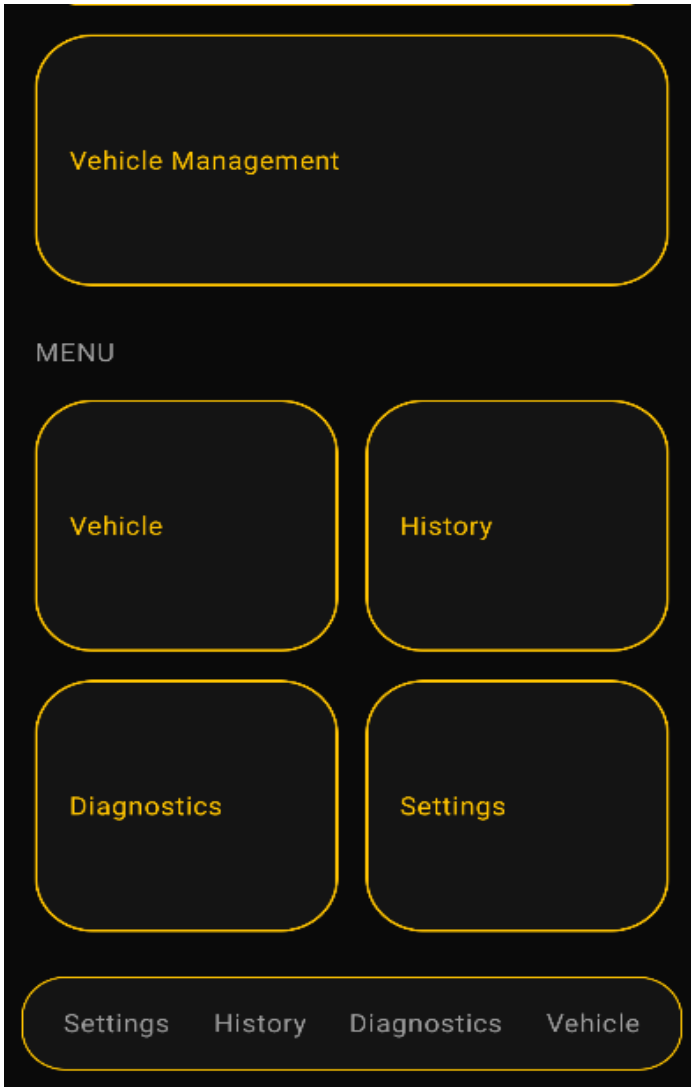

Рисунок 3.8 - Головний екран системи

3.4.3 Реалізація екрана телеметричних параметрів

Екран телеметричних параметрів використовується для відображення основних показників транспортного засобу у режимі реального часу. На даному екрані користувач може переглядати швидкість руху, температуру двигуна, рівень пального, оберти двигуна та напругу акумулятора [32].

Для зручності сприйняття інформації параметри автомобіля були розділені на окремі інформаційні блоки. Подібний підхід дозволяє спростити перегляд телеметричних даних та забезпечує швидкий доступ до основних показників автомобіля.

На рисунку 3.9 наведено приклад екрана телеметричних параметрів.

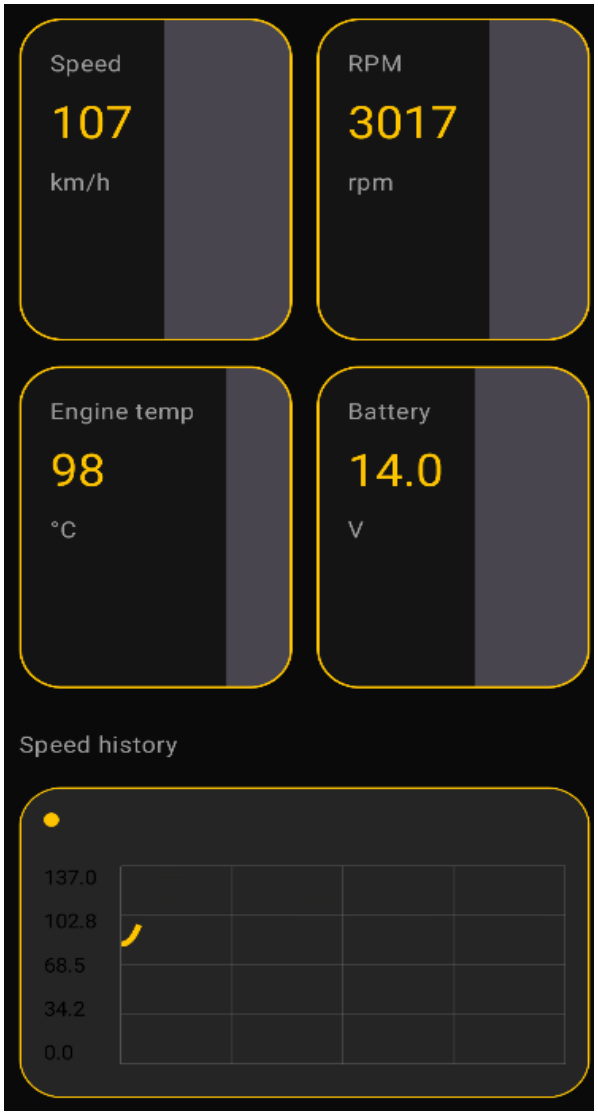

Рисунок 3.9 - Екран телеметричних параметрів

Лістинг 3.16 - Отримання телеметричних даних від серверу

```
viewModelScope.launch {

    val telemetry =
        repository.getTelemetry()

    _state.value =
        _state.value.copy(
            speed = telemetry.speed,
            rpm = telemetry.rpm,
            engineTemp = telemetry.engineTemp,
            batteryVoltage = telemetry.batteryVoltage
        )
}
```

У наведеному лістингу реалізовано отримання телеметричних даних від серверної частини системи. Після виконання запиту до API отримані значення передаються до об’єкта стану ViewModel. Зміна стану автоматично викликає оновлення компонентів Jetpack Compose, що забезпечує відображення актуальної інформації про стан автомобіля на екрані користувача [35].

Приклад відображення телеметричних параметрів наведено у лістингу 3.17.

Лістинг 3.17 - Відображення телеметричних параметрів автомобіля

```
TelemetryCard(
    title = "Speed",
    value = "${state.speed} km/h"
)
TelemetryCard(
    title = "RPM",
    value = "${state.rpm}"
)
TelemetryCard(
    title = "Battery",
    value = "${state.batteryVoltage} V"
)
```

У наведеному лістингу реалізовано відображення основних телеметричних параметрів автомобіля у користувацькому інтерфейсі застосунку. Для відображення параметрів використовуються окремі інформаційні картки, які дозволяють користувачу швидко отримувати інформацію про поточний стан транспортного засобу.

3.4.4 Реалізація екрана статистики

Екран статистики використовується для перегляду історії телеметричних параметрів автомобіля та аналізу зміни показників транспортного засобу протягом певного періоду часу.

На даному екрані можуть відображатись попередні значення параметрів автомобіля, інформація про стан транспортного засобу та результати роботи системи моніторингу.

На рисунку 3.10 наведено приклад екрана статистики мобільного застосунку.

Рисунок 3.10 - Екран статистики системи

Для побудови екрана статистики використовувались стандартні компоненти Jetpack Compose та елементи списків, які дозволяють відображати інформацію у структурованому вигляді.

Приклад реалізації списку статистичних даних наведено у лістингу 3.18.

Лістинг 3.18 - Реалізація списку статистичних даних

```
LazyColumn(
```

```
    verticalArrangement =
        Arrangement.spacedBy(12.dp)
) {
    items(historyList) { item ->
        HistoryCard(item)
    }
}
```

У наведеному лістингу використовується компонент LazyColumn, який дозволяє відображати список статистичних даних у мобільному застосунку. Компонент автоматично оптимізує відображення великої кількості елементів та забезпечує коректну роботу прокручування списку.

Під час реалізації користувацького інтерфейсу також було враховано необхідність забезпечення адаптивності застосунку для різних мобільних пристроїв. Використання компонентів Jetpack Compose та Material Design дозволило реалізувати сучасний інтерфейс мобільного застосунку та забезпечити зручну взаємодію користувача із системою моніторингу стану автомобіля.

3.5 Висновки до розділу

Реалізація мобільного застосунку дозволила забезпечити зручну взаємодію користувача із системою та організувати відображення телеметричних параметрів автомобіля у режимі реального часу. Під час розробки було створено основні екрани застосунку, реалізовано механізми навігації, авторизації користувачів та відображення інформації про транспортний засіб. Використання архітектури MVVM дозволило розділити логіку роботи застосунку та інтерфейс користувача, що позитивно вплинуло на структуру програмного коду та його подальшу підтримку.

Для забезпечення обміну інформацією між окремими компонентами системи було реалізовано серверну частину на основі платформи Node.js. Сервер забезпечує обробку HTTP-запитів, взаємодію з базою даних PostgreSQL та

передачу інформації між мобільним застосунком і сховищем даних. Використання REST API та формату JSON дозволило реалізувати стандартизований механізм передачі інформації та забезпечити сумісність між програмними компонентами системи.

Окремим результатом реалізації стало створення структури бази даних для збереження інформації про користувачів, транспортні засоби та телеметричні параметри. Реалізована структура забезпечує цілісність даних, підтримує зв’язки між основними сутностями та дозволяє накопичувати історію показників автомобіля для їх подальшого аналізу. У результаті була забезпечена повна програмна реалізація процесу отримання, передачі, збереження та відображення телеметричних даних у межах єдиної інформаційної системи.

Таким чином, у третьому розділі було реалізовано всі основні програмні компоненти системи моніторингу стану автомобіля та забезпечено їхню взаємодію між собою. Отримані результати дозволяють перейти до проведення випробувань розробленого програмного забезпечення та оцінювання його працездатності в умовах, наближених до реальної експлуатації.

4 ВИПРОБУВАННЯ ТА ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ

Після завершення реалізації комп’ютерної системи моніторингу стану автомобіля було проведено випробування основних функціональних можливостей програмного забезпечення. Метою випробувань є перевірка працездатності розробленої системи, коректності взаємодії між мобільним застосунком, серверною частиною та базою даних, а також підтвердження можливості отримання, передачі та збереження телеметричних даних транспортного засобу.

Під час випробувань перевірялись процес підключення автомобіля через інтерфейс OBD-II, отримання телеметричних параметрів, передача інформації до серверної частини, збереження даних у базі даних PostgreSQL та їх подальше

відображення у мобільному застосунку. Окрему увагу було приділено перевірці сценарію роботи користувача із системою та взаємодії між усіма основними компонентами програмного забезпечення.

4.1 Випробування підключення автомобіля та отримання телеметричних даних

Однією з основних функцій розробленої системи є отримання телеметричних даних від транспортного засобу та їх подальше відображення у мобільному застосунку. Метою даного випробування є перевірка працездатності механізму підключення до автомобіля через інтерфейс OBD-II та отримання основних параметрів роботи транспортного засобу.

Для забезпечення взаємодії між автомобілем та мобільним застосунком використовується адаптер ELM327, який підключається до діагностичного роз'єму OBD-II та забезпечує передачу телеметричних даних через бездротовий інтерфейс Bluetooth. Після запуску застосунку виконується пошук доступного адаптера та підготовка до встановлення з'єднання з транспортним засобом.

На рисунку 4.1 наведено процес пошуку OBD-II адаптера мобільним застосунком.

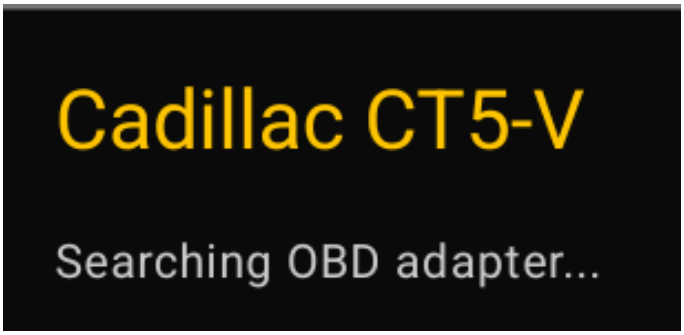

Рисунок 4.1 - Пошук OBD-II адаптера

Під час проведення випробування система успішно виявила доступний адаптер та виконала процедуру встановлення з’єднання. Після завершення підключення застосунок отримав можливість надсилати запити до електронного блоку керування автомобіля та отримувати необхідні телеметричні параметри.

Результат успішного підключення до OBD-II адаптера наведено на рисунку 4.2.

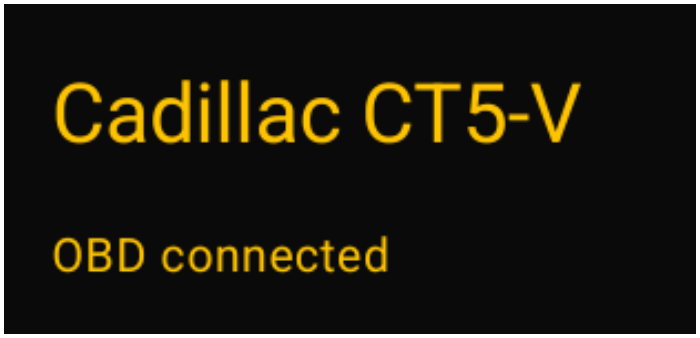

Рисунок 4.2 - Підключення до OBD-II адаптера

Після встановлення з’єднання було виконано отримання телеметричних даних транспортного засобу. У процесі випробування система отримувала основні параметри роботи автомобіля, зокрема швидкість руху, оберти двигуна, температуру двигуна та напругу акумулятора. Отримані дані автоматично оброблялися мобільним застосунком та відображалися у вигляді окремих інформаційних карток користувацького інтерфейсу.

Приклад відображення телеметричних параметрів наведено на рисунку 4.3.

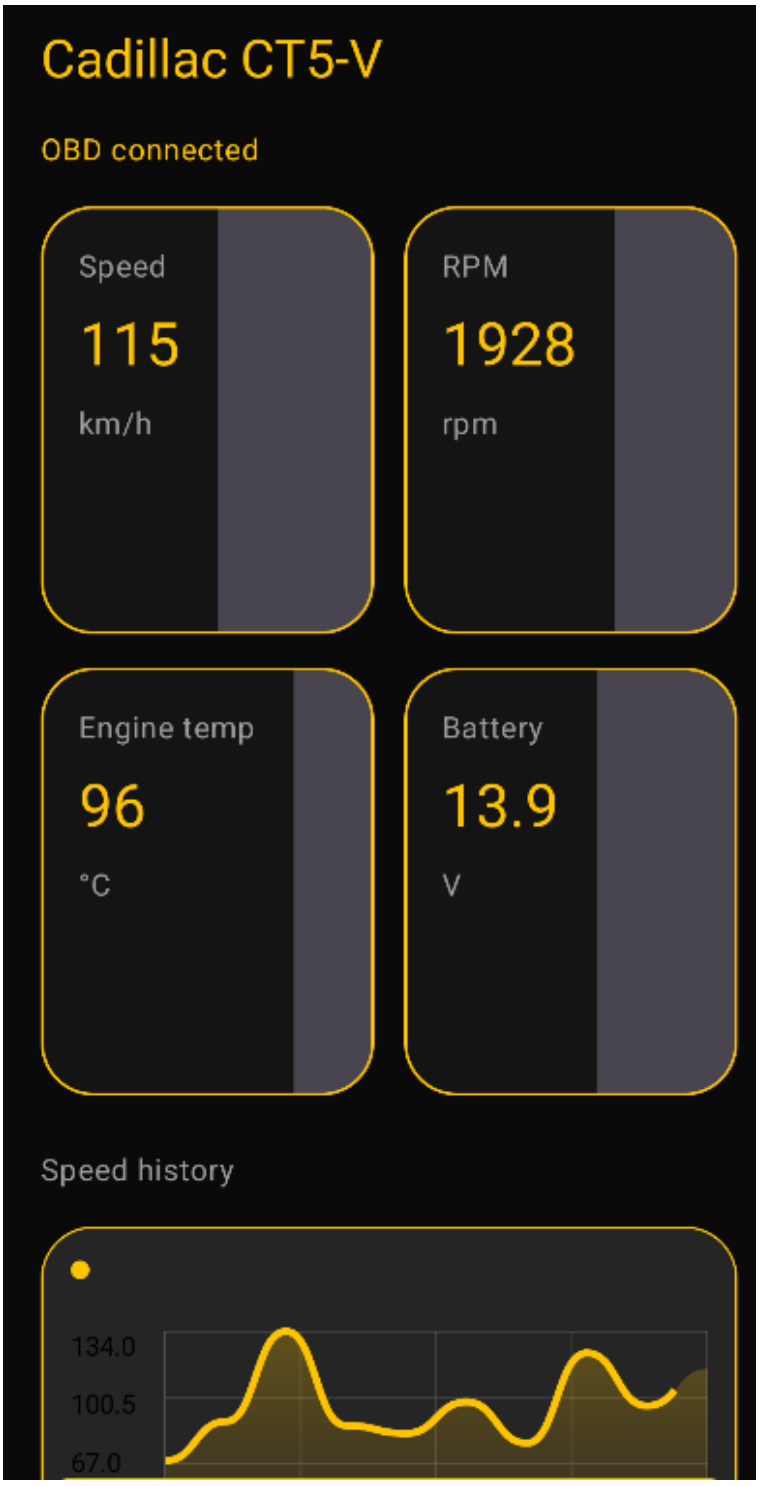

Рисунок 4.3 - Відображення телеметричних даних автомобіля

Під час випробування було встановлено, що процес отримання телеметричних даних виконується коректно, а оновлення інформації на екрані застосунку відбувається автоматично після надходження нових значень від автомобіля. Отримані результати підтвердили працездатність механізму взаємодії між транспортним засобом, OBD-II адаптером та мобільним застосунком і

дозволили перейти до перевірки передачі та обробки даних серверною частиною системи.

4.2 Випробування передачі та обробки телеметричних даних

Після отримання телеметричних параметрів від транспортного засобу важливим етапом роботи системи є передача інформації до серверної частини та її подальша обробка. Метою даного випробування є перевірка коректності роботи механізмів обміну даними між мобільним застосунком, сервером та базою даних.

Передача інформації у системі здійснюється за допомогою REST API. Мобільний застосунок формує HTTP-запити та передає телеметричні параметри до серверної частини у форматі JSON. Сервер виконує обробку отриманої інформації, перевіряє коректність даних та забезпечує їх подальше збереження у базі даних PostgreSQL.

Для перевірки роботи серверної частини було виконано тестові API-запити за допомогою середовища Postman. У процесі випробування сервер успішно приймав запити від клієнтської частини та повертав коректні відповіді відповідно до реалізованих маршрутів API.

Приклад виконання запиту до серверної частини наведено на рисунку 4.4 та 4.5.

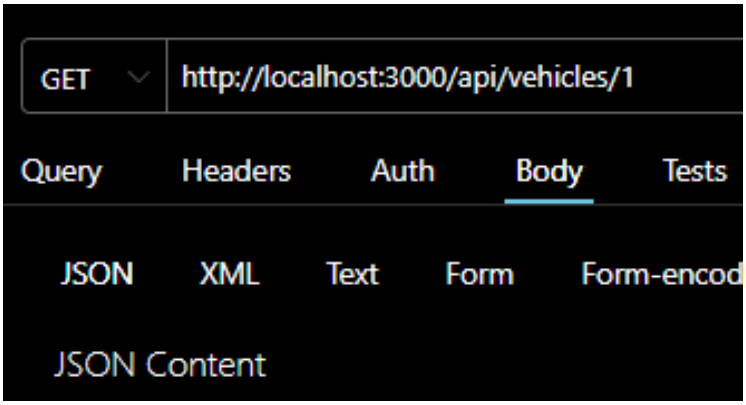

Рисунок 4.4 - Відправка запиту REST API за допомогою Postman

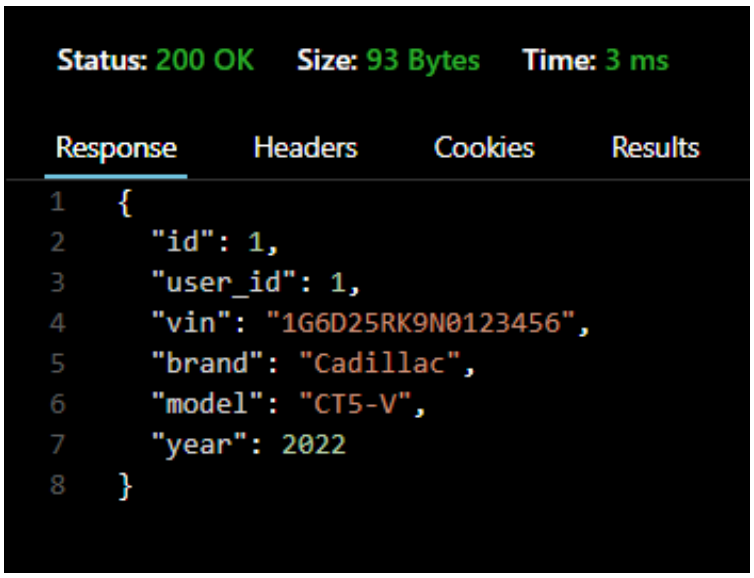

Рисунок 4.5 - Отриманий результат запиту REST API за допомогою Postman

Після отримання запиту сервер виконує необхідну обробку інформації та взаємодіє із базою даних. Для перевірки коректності збереження інформації було виконано перегляд записів у базі даних PostgreSQL. Результати випробування підтвердили успішне збереження інформації про користувачів, транспортні засоби та телеметричні параметри.

Приклад збережених даних у базі даних наведено на рисунку 4.6.

Data Output | Messages | Notifications

	id [PK] integer	vehicle_id integer	speed integer	rpm integer	engine_temp integer	battery_voltage real	created_at timestamp without time zone
1	1	1	120	3400	95	13.7	2026-05-23 16:20:08.067667
2	2	1	60	2503	98	14.310624	2026-05-24 15:03:48.933298
3	3	1	94	2790	89	13.852042	2026-05-24 15:03:50.577171
4	4	1	83	3974	95	13.890116	2026-05-24 15:03:52.093946
5	5	1	82	3182	97	13.347445	2026-05-24 15:03:58.14781
6	6	1	77	2670	95	13.513679	2026-05-24 15:03:59.697157
7	7	1	138	2654	90	13.509066	2026-05-24 15:04:01.205207

Рисунок 4.6 - Збереження інформації у базі даних PostgreSQL

Після обробки запиту сервер формує відповідь у форматі JSON та передає її клієнтській частині системи. Отримані дані використовуються мобільним

застосунком для оновлення інформації про стан автомобіля та відображення актуальних параметрів користувачу.

Для перевірки відповіді серверної частини було виконано GET-запит до API. Сервер повернув дані у форматі JSON, які надалі використовуються мобільним застосунком для відображення інформації про автомобіль або його телеметричні параметри.

Приклад відповіді серверної частини наведено на рисунку 4.7.

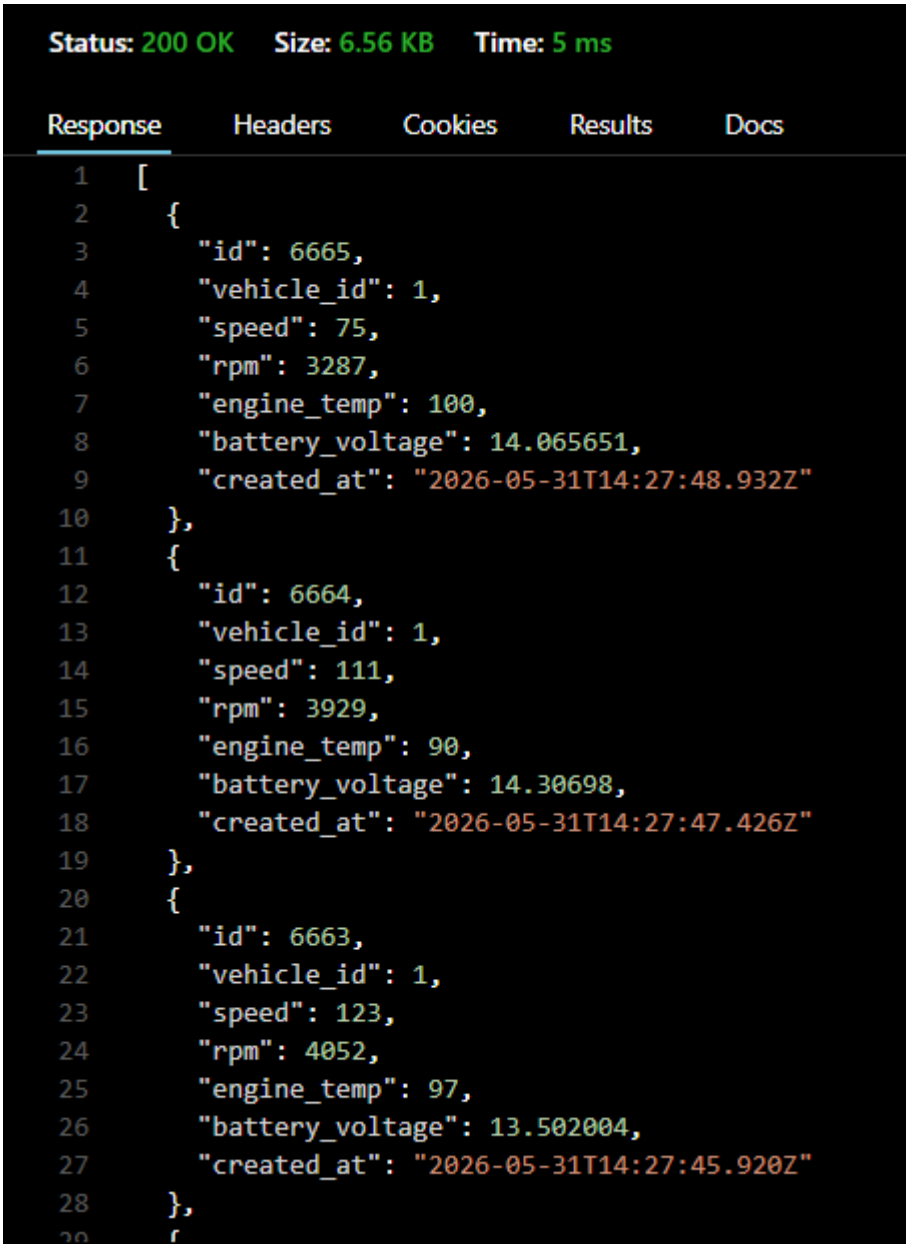

Рисунок 4.7 - Приклад відповіді серверної частини системи

У результаті проведеного випробування було підтверджено коректність роботи механізмів передачі, обробки та збереження даних. Серверна частина успішно приймає запити від клієнтського застосунку, забезпечує взаємодію із базою даних та повертає результати обробки у стандартизованому форматі JSON. Отримані результати підтверджують працездатність клієнт-серверної архітектури розробленої системи та можливість її використання для обробки телеметричної інформації автомобіля.

4.3 Випробування користувацького інтерфейсу

Основною задачею даного випробування є перевірка коректності роботи користувацького інтерфейсу мобільного застосунку та зручності взаємодії користувача із функціональними можливостями системи. Під час випробування перевірялась робота основних екранів застосунку, коректність відображення телеметричних даних та навігація між окремими компонентами програмного забезпечення. [35].

Після запуску мобільного застосунку користувач переходить до екрана авторизації. На даному екрані розташовані поля введення логіна та пароля, а також кнопка входу до системи та реєстрації.

На рисунку 4.8 наведено екран авторизації мобільного застосунку.

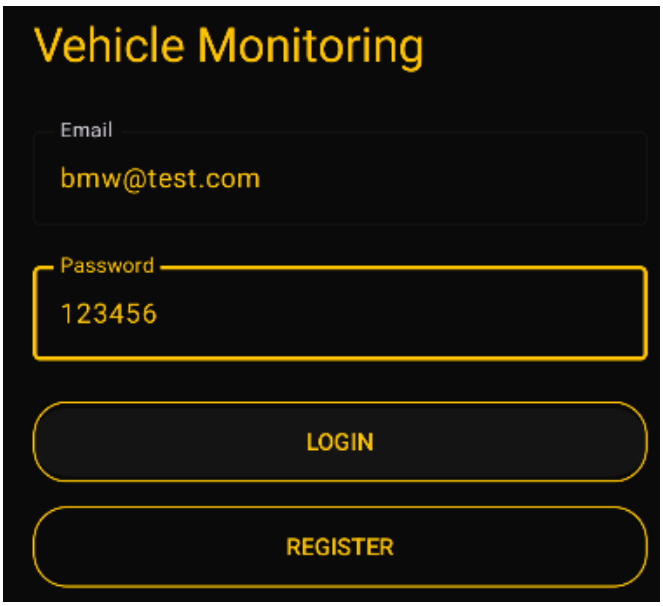

Рисунок 4.8 - Екран авторизації мобільного застосунку

Під час випробування було встановлено, що всі елементи інтерфейсу коректно відображаються на мобільному пристрої, а процес авторизації виконується без помилок.

Після успішного входу до системи користувач переходить до головного екрана застосунку. Даний екран використовується для відображення основної інформації про транспортний засіб та поточних телеметричних параметрів.

На рисунку 4.9 наведено головний екран системи моніторингу стану автомобіля.

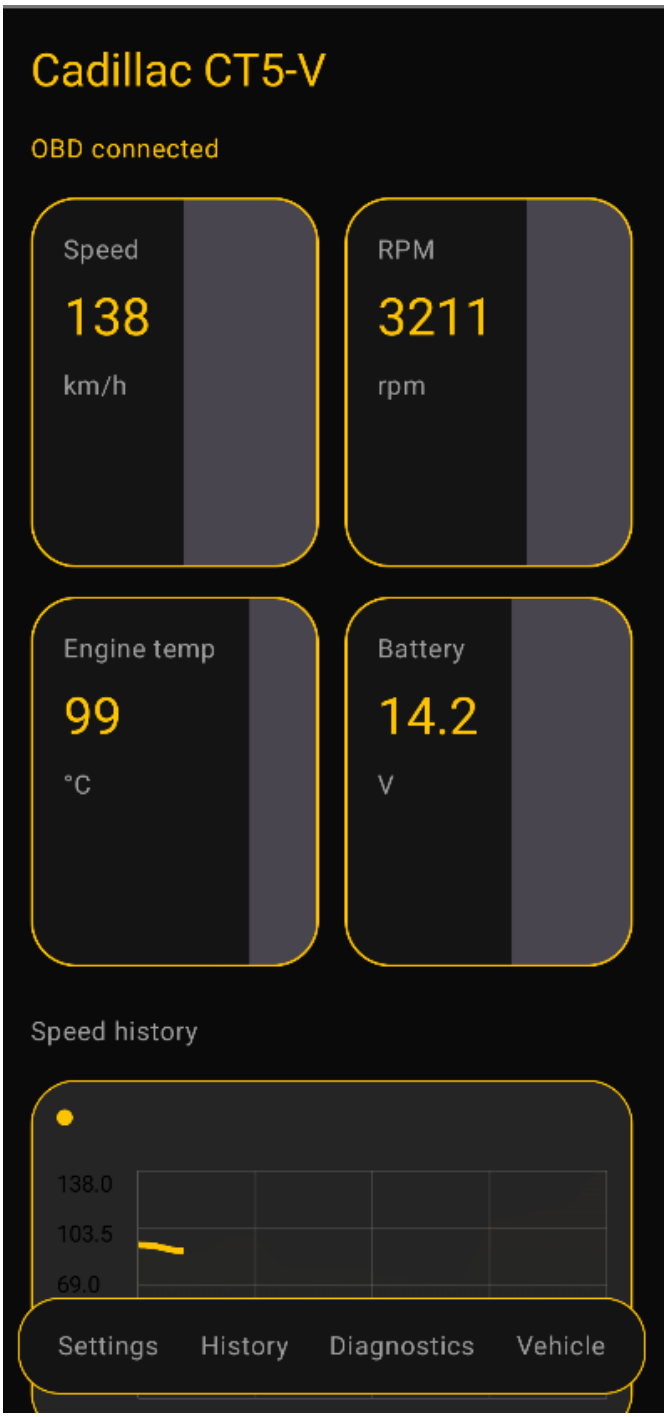

Рисунок 4.9 - Головний екран системи моніторингу

Головний екран містить інформаційні картки із основними параметрами транспортного засобу, що забезпечує швидкий доступ до найбільш важливої інформації під час роботи із системою.

Для аналізу накопичених показників у застосунку реалізовано окремий екран статистики. На даному екрані користувач отримує можливість переглядати

історію телеметричних даних та аналізувати зміну параметрів автомобіля протягом певного періоду часу.

На рисунку 4.10 наведено екран статистики мобільного застосунку.

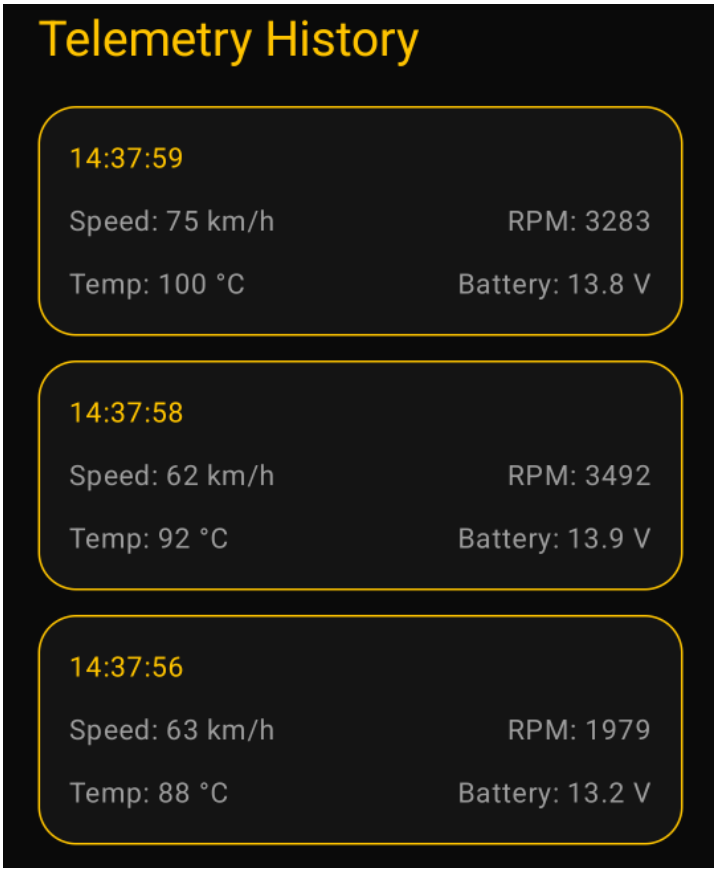

Рисунок 4.10 - Екран статистики мобільного застосунку

Під час випробування було встановлено, що статистична інформація коректно відображається у застосунку, а елементи інтерфейсу забезпечують зручне сприйняття отриманих даних.

Для забезпечення швидкого доступу до функціональних можливостей системи у застосунку реалізовано механізм навігації між основними екранами. Під час випробування перевірялась коректність переходів між екранами та стабільність роботи навігаційних компонентів.

На рисунку 4.11 наведено приклад навігації між основними екранами системи.

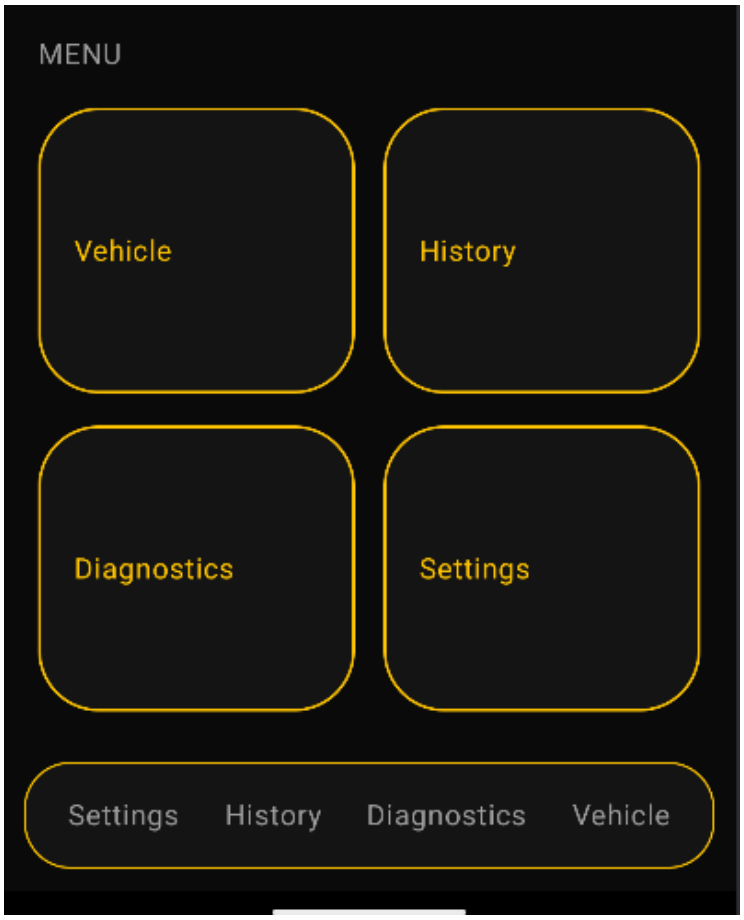

Рисунок 4.11 - Навігація між екранами мобільного застосунку

Під час випробування користувацького інтерфейсу не було виявлено помилок відображення інформації або порушень роботи окремих компонентів системи. Мобільний застосунок коректно відображає інформацію про стан автомобіля та забезпечує зручну взаємодію користувача із системою моніторингу стану автомобіля.

4.4 Випробування сценарію моніторингу нового автомобіля

Завершальним етапом випробувань стала перевірка повного сценарію моніторингу нового транспортного засобу. Метою даного випробування є перевірка взаємодії всіх компонентів системи під час виконання основних

функцій, починаючи від додавання автомобіля та завершуючи переглядом статистичних даних.

На першому етапі користувач виконує реєстрацію у мобільному застосунку за допомогою введення електронної пошти та вигаданого паролю. Після успішного входу система надає доступ до основних функціональних можливостей програмного забезпечення.

Екран авторизації наведено на рисунку 4.12.

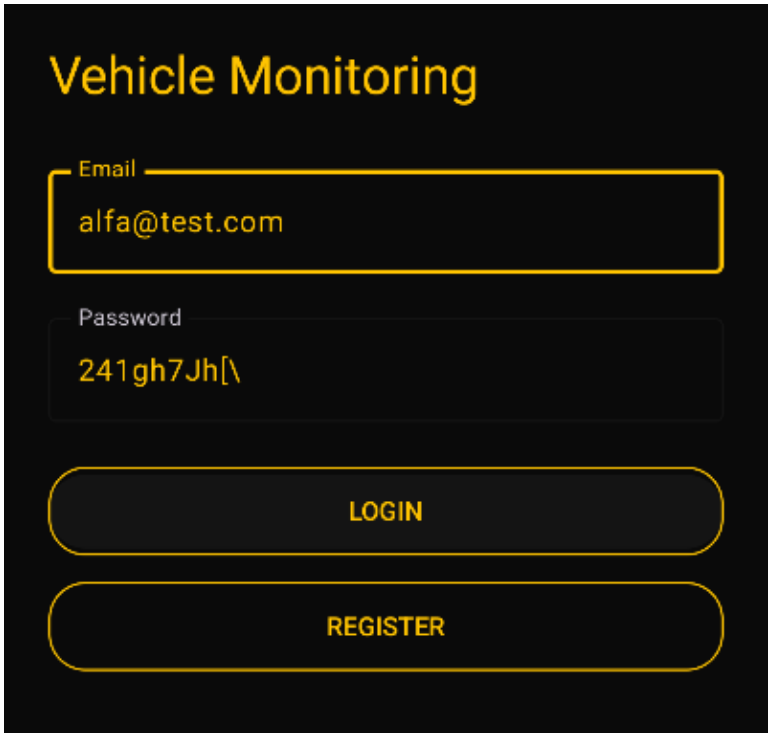

Рисунок 4.12 - Реєстрація нового користувача у системі

Після входу до системи користувач виконує додавання нового автомобіля. На цьому етапі в систему вносяться основні відомості про транспортний засіб, необхідні для подальшого моніторингу та збереження інформації.

Процес додавання автомобіля наведено на рисунку 4.13.

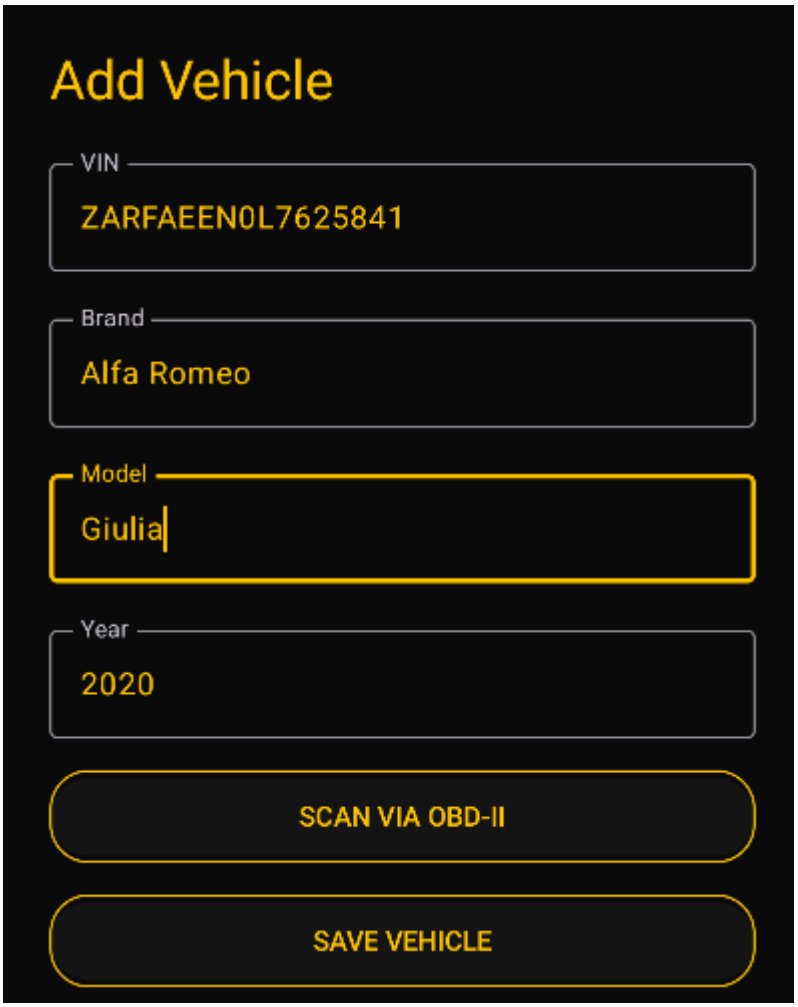

Рисунок 4.13 - Додавання нового автомобіля

Після додавання транспортного засобу інформація про автомобіль передається до серверної частини та зберігається у базі даних PostgreSQL. Наявність нового запису у таблиці автомобілів підтверджує успішне виконання операції додавання транспортного засобу до системи.

Приклад збереження інформації про автомобіль наведено на рисунку 4.14.

id [PK] integer	user_id integer	vin character varying (50)	brand character varying (100)	model character varying (100)	year integer
1	1	1G6D25RK9N0123456	Cadillac	CT5-V	2022
2	3	1G6D25RK9N014563	BMW	M3	2015
3	4	1G6D25RK9N524365	Audi	Q7	2026
6	7	ZARFAEEN0L7625841	Alfa Romeo	Giulia	2020

Рисунок 4.14 - Збереження інформації про автомобіль у базі даних

Після завершення додавання транспортного засобу система починає отримувати телеметричні параметри автомобіля. Отримана інформація відображається у мобільному застосунку та використовується для подальшої передачі на сервер.

Приклад отриманих телеметричних параметрів наведено на рисунку 4.15.

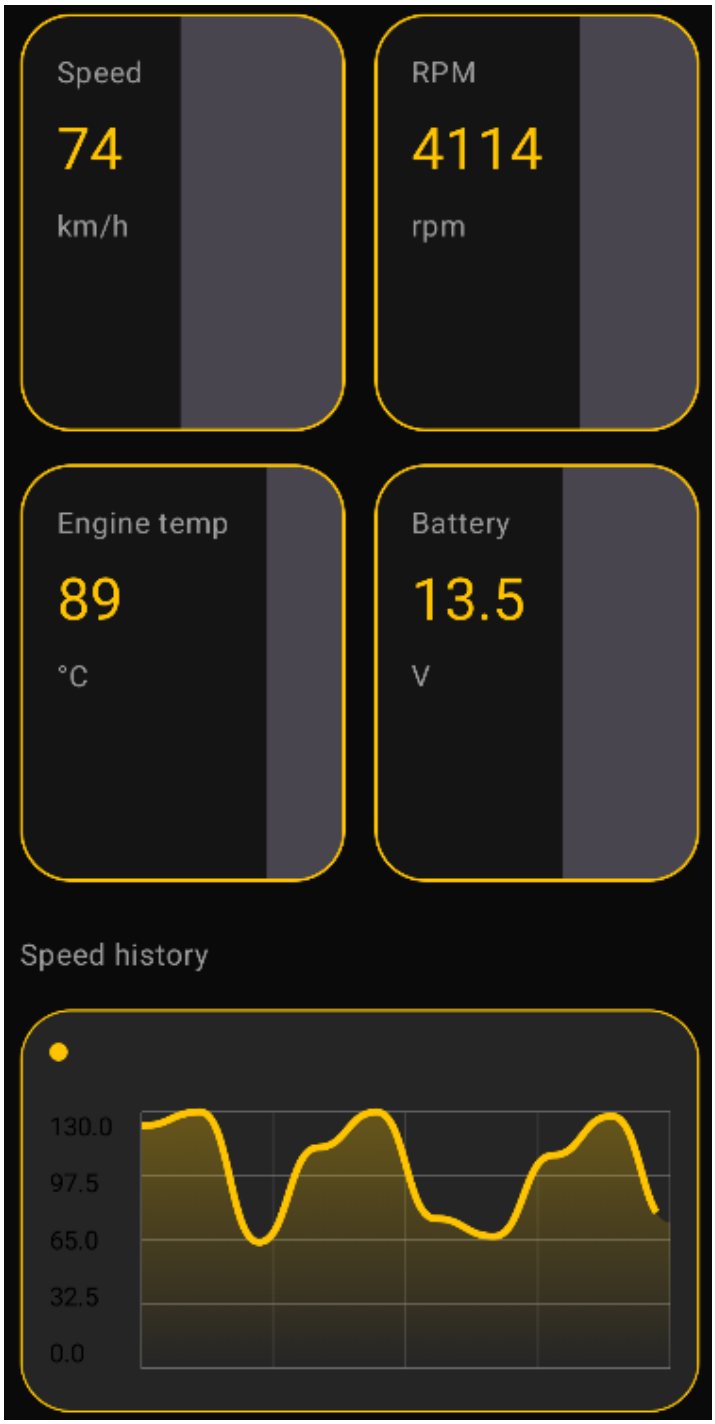

Рисунок 4.15 - Відображення телеметричних параметрів автомобіля

Під час роботи системи телеметричні дані передаються до серверної частини та зберігаються у базі даних PostgreSQL. Це дозволяє накопичувати історію показників та використовувати її для подальшого аналізу.

Приклад збережених записів наведено на рисунку 4.16.

id [PK] integer	vehicle_id integer	speed integer	rpm integer	engine_temp integer	battery_voltage real	created_at timestamp without time zone
1	1	120	3400	95	13.7	2026-05-23 16:20:08.067667
2	1	60	2503	98	14.310624	2026-05-24 15:03:48.933298
3	1	94	2790	89	13.852042	2026-05-24 15:03:50.577171
4	1	83	3974	95	13.890116	2026-05-24 15:03:52.093946
5	1	82	3182	97	13.347445	2026-05-24 15:03:58.14781
6	1	77	2670	95	13.513679	2026-05-24 15:03:59.697157
7	1	138	2654	90	13.509066	2026-05-24 15:04:01.205207
8	1	105	3229	97	13.249588	2026-05-24 15:05:15.032107
9	1	68	3702	88	13.988068	2026-05-24 15:05:16.576048

Рисунок 4.16 - Збереження телеметричних даних у базі даних

Для аналізу отриманої інформації користувач може перейти до екрана статистики, де відображаються накопичені параметри транспортного засобу. Використання статистичних даних дозволяє оцінювати зміни показників автомобіля та контролювати його технічний стан протягом певного періоду часу.

Екран статистики наведено на рисунку 4.17.

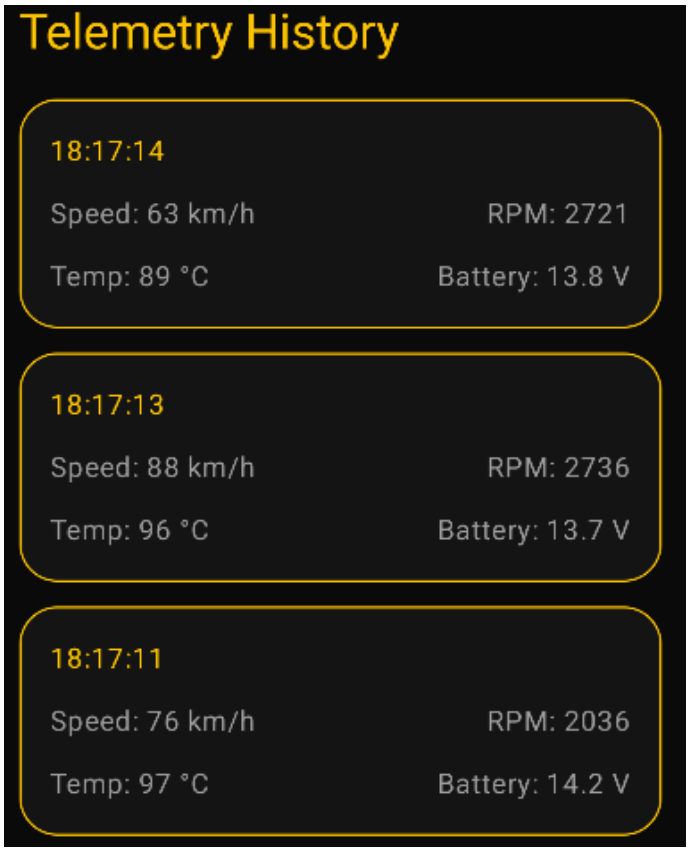

Рисунок 4.17 - Перегляд статистичних даних автомобіля

У результаті проведеного випробування всі етапи сценарію були виконані успішно. Система забезпечила коректне додавання транспортного засобу, отримання телеметричних параметрів, передачу даних до серверної частини, збереження інформації у базі даних та її подальше відображення у вигляді статистичних даних. Отримані результати підтверджують працездатність усіх основних компонентів розробленої системи моніторингу стану автомобіля.

4.5 Висновки до розділу

Перевірка працездатності комп’ютерної системи моніторингу стану автомобіля дозволила підтвердити коректність функціонування основних компонентів розробленого програмного забезпечення. Особливу увагу було приділено перевірці процесу отримання телеметричних даних, їх передачі між

окремими компонентами системи та подальшому відображенню у мобільному застосунку.

Проведені випробування підтвердили можливість отримання інформації про стан транспортного засобу через інтерфейс OBD-II із використанням адаптера ELM327. У процесі випробувань було перевірено коректність формування та обробки телеметричних параметрів, зокрема швидкості руху автомобіля, обертів двигуна, температури охолоджувальної рідини та напруги бортової мережі. Отримані результати підтвердили можливість використання розробленої системи для збору та первинної обробки інформації про технічний стан транспортного засобу.

Окремо було виконано перевірку серверної частини системи та механізмів взаємодії із базою даних. У ході випробувань підтверджено коректність роботи API-запитів, передачі інформації у форматі JSON та збереження даних у базі даних PostgreSQL. Проведена перевірка також показала коректне виконання операцій додавання користувачів, реєстрації транспортних засобів та накопичення телеметричних параметрів для подальшого аналізу.

Перевірка користувацького сценарію роботи системи підтвердила можливість виконання повного циклу взаємодії із програмним забезпеченням. Користувач отримує можливість зареєструвати обліковий запис, додати транспортний засіб, переглядати поточні телеметричні параметри та аналізувати накопичені статистичні дані. Усі етапи сценарію були виконані без критичних помилок, а результати випробувань підтвердили працездатність розробленої комп’ютерної системи моніторингу стану автомобіля та відповідність поставленим вимогам.

ВИСНОВКИ

У результаті виконання дипломної роботи було досягнуто поставленої мети та розроблено комп’ютерну систему моніторингу стану автомобіля, яка забезпечує отримання, обробку та відображення телеметричних параметрів транспортного засобу за допомогою мобільного застосунку.

Під час виконання першого розділу було проведено аналіз предметної області, розглянуто існуючі аналоги систем моніторингу автомобілів та визначено основні функціональні вимоги до програмного забезпечення. На основі проведеного аналізу у другому розділі було сформовано структуру системи, обрано клієнт-серверну архітектуру та спроєктовано базу даних і механізм збору телеметричних даних.

У межах третього розділу було реалізовано мобільний застосунок, серверну частину та базу даних системи. Для забезпечення взаємодії між окремими компонентами програмного забезпечення реалізовано API для передачі інформації між клієнтською та серверною частинами системи, а також механізми отримання телеметричних параметрів через адаптер ELM327.

У четвертому розділі проведено випробування розробленої системи, які підтвердили коректність роботи основних функцій, передачі даних між компонентами та збереження інформації у базі даних.

Таким чином, поставлена мета дипломної роботи була досягнута. Розроблена комп’ютерна система моніторингу стану автомобіля може використовуватись як основа для подальшого розвитку систем автомобільної телеметрії та розширення функціональних можливостей мобільних застосунків для контролю технічного стану транспортних засобів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Automotive electronics. Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Automotive_electronics (дата звернення: 14.04.2026).

2. Vehicle telematics. Geotab [Електронний ресурс]. URL: https://www.geotab.com/blog/vehicle-telematics/ (дата звернення: 15.04.2026).

3. Mobile app. Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Mobile_app (дата звернення: 16.04.2026).

4. Client-server model. TechTarget [Електронний ресурс]. URL: https://www.techtarget.com/searchnetworking/definition/client-server (дата звернення: 17.04.2026).

5. Fleet telematics. Verizon Connect [Електронний ресурс]. URL: https://www.verizonconnect.com/resources/article/what-is-telematics/ (дата звернення: 18.04.2026).

6. On-board diagnostics. Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/On-board_diagnostics (дата звернення: 19.04.2026).

7. OBD-II PIDs. Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/OBD-II_PIDs (дата звернення: 20.04.2026).

8. Client–server architecture. MDN Web Docs [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/Client-Server_overview (дата звернення: 21.04.2026).

9. Android Developers Documentation [Електронний ресурс]. URL: https://developer.android.com/docs (дата звернення: 22.04.2026).

10. REST API Tutorial [Електронний ресурс]. URL: https://restfulapi.net/ (дата звернення: 23.04.2026).

11. Android Studio Documentation [Електронний ресурс]. URL: https://developer.android.com/studio/intro (дата звернення: 24.04.2026).

12. Kotlin Documentation [Електронний ресурс]. URL: https://kotlinlang.org/docs/home.html (дата звернення: 25.04.2026).

13. SQL Tutorial. W3Schools [Електронний ресурс]. URL: https://www.w3schools.com/sql/ (дата звернення: 26.04.2026).

14. Android Developers. App architecture [Електронний ресурс]. URL: https://developer.android.com/topic/architecture (дата звернення: 27.04.2026).

15. Software requirements. IBM Documentation [Електронний ресурс]. URL: https://www.ibm.com/topics/software-requirements (дата звернення: 28.04.2026).

16. Non-functional requirements. AltexSoft [Електронний ресурс]. URL: https://www.altexsoft.com/blog/non-functional-requirements/ (дата звернення: 29.04.2026).

17. REST API Tutorial [Електронний ресурс]. URL: https://restfulapi.net/ (дата звернення: 30.04.2026).

18. MDN Web Docs. HTTP Overview [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (дата звернення: 01.05.2026).

19. Client-server model. Cloudflare [Електронний ресурс]. URL: https://www.cloudflare.com/learning/serverless/glossary/client-server-model/ (дата звернення: 02.05.2026).

20. PostgreSQL Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/ (дата звернення: 03.05.2026).

21. JSON Official Website [Електронний ресурс]. URL: https://www.json.org/json-en.html (дата звернення: 04.05.2026).

22. Functional requirements. IBM Documentation [Електронний ресурс]. URL: https://www.ibm.com/topics/functional-requirements (дата звернення: 05.05.2026).

23. Android Developers. Security best practices [Електронний ресурс]. URL: https://developer.android.com/topic/security/best-practices (дата звернення: 06.05.2026).

24. SAE J1979 Standard PIDs. GitHub [Електронний ресурс]. URL: https://github.com/OBDb/SAEJ1979 (дата звернення: 07.05.2026).

25. Diagnostic Trouble Codes. Wikipedia [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/OBD-II_PIDs#Service_03_(no_PID_required)_-_Show_stored_Diagnostic_Trouble_Codes (дата звернення: 08.05.2026).

26. Material Design 3 [Електронний ресурс]. URL: https://m3.material.io/ (дата звернення: 09.05.2026).

27. Visual Paradigm. What is Use Case Diagram? [Електронний ресурс]. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 10.05.2026).

28. Visual Paradigm. How to Draw Use Case Diagram? [Електронний ресурс]. URL: https://www.visual-paradigm.com/support/documents/vpuserguide/94/2575/6362_drawinguseca.html (дата звернення: 11.05.2026).

29. Vehicle diagnostics. Geotab [Електронний ресурс]. URL: https://www.geotab.com/blog/obd-ii/ (дата звернення: 12.05.2026).

30. SQLite Documentation [Електронний ресурс]. URL: https://www.sqlite.org/docs.html (дата звернення: 13.05.2026).

31. Visual Paradigm. Entity Relationship Diagram [Електронний ресурс]. URL: https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/ (дата звернення: 14.05.2026).

32. OBD-II Standard PIDs. Total Car Diagnostics [Електронний ресурс]. URL: https://www.totalcardiagnostics.com/support/Knowledgebase/Article/View/104/0/obd2-pids-for-programmers-technical (дата звернення: 15.05.2026).

33. Android Developers. Build a user interface [Електронний ресурс]. URL: https://developer.android.com/develop/ui (дата звернення: 16.05.2026).

34. Android Developers. Support different screen sizes [Електронний ресурс]. URL: https://developer.android.com/develop/ui/compose/layouts/adaptive/support-different-screen-sizes (дата звернення: 17.05.2026).

35. Jetpack Compose Documentation [Електронний ресурс]. URL: https://developer.android.com/jetpack/compose (дата звернення: 18.05.2026).

36. MDN Web Docs. Working with JSON [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON (дата звернення: 19.05.2026).

37. MDN Web Docs. HTTP request methods [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (дата звернення: 20.05.2026).

38. Android Developers. Navigation [Електронний ресурс]. URL: https://developer.android.com/guide/navigation (дата звернення: 21.05.2026).

39. Android Developers. ViewModel overview [Електронний ресурс]. URL: https://developer.android.com/topic/libraries/architecture/viewmodel (дата звернення: 22.05.2026).

40. Retrofit Documentation [Електронний ресурс]. URL: https://square.github.io/retrofit/ (дата звернення: 27.05.2026).

ДОДАТОК А
ЛІСТИНГИ ПРОГРАМНОЇ ЧАСТИНИ

Лістинг А.1 - Код головного файлу серверної частини

```
const express = require("express")
const cors = require("cors")
require("dotenv").config()
const pool = require("./database/db")
const authRoutes =
    require("./routes/authRoutes")
const vehicleRoutes =
    require("./routes/vehicleRoutes")
const app = express()
const telemetryRoutes =
    require("./routes/telemetryRoutes")

app.use(cors())
app.use(express.json())
app.use("/api/auth", authRoutes)
app.use("/api/vehicles", vehicleRoutes)
app.use("/api/telemetry", telemetryRoutes)

pool.connect()
    .then(() => {

        console.log("PostgreSQL connected")
    })
    .catch((err) => {

        console.log(err.message)
    })

app.get("/", (req, res) => {

    res.json({

        message: "Vehicle Monitoring API"
    })
})

const PORT = process.env.PORT || 3000
app.get("/api/status", (req, res) => {

    res.json({

        status: "online"
    })
})
app.listen(PORT, () => {
```

```
    console.log(`Server running on port ${PORT}`)
})
```

Лістинг А.2 - Код підключення до бази даних PostgreSQL

```
const { Pool } = require("pg")

const pool = new Pool({

    host: "localhost",

    port: 5432,

    user: "postgres",

    password: "Foldonp2405.",

    database: "vehicle_monitoring"
})

module.exports = pool
```

Лістинг А.3 - Код маршруту авторизації користувача

```
const express = require("express")

const router = express.Router()

const {
    register,
    login
} = require("../controllers/authController")

router.post("/register", register)
router.post("/login", login)

module.exports = router
```

Лістинг А.4 - Код контролера авторизації користувача

```
const pool = require("../database/db")
const bcrypt = require("bcryptjs")
const jwt = require("jsonwebtoken")

const register = async (req, res) => {

    try {

        const { email, password } = req.body
```

```
        const hashedPassword =
            await bcrypt.hash(password, 10)

        const result = await pool.query(

            `
            INSERT INTO users (email, password)

            VALUES ($1, $2)

            RETURNING id, email
            `,

            [email, hashedPassword]
        )

        res.status(201).json({

            message: "User created",

            user: result.rows[0]
        })

    } catch (err) {

        console.log(err.message)

        res.status(500).json({

            error: "Server error"
        })
    }
}
const login = async (req, res) => {

    try {

        const { email, password } = req.body

        const result = await pool.query(

            `
            SELECT * FROM users

            WHERE email = $1
            `,

            [email]
        )

        if(result.rows.length === 0) {

            return res.status(401).json({
```

```
                error: "Invalid email"
            })
        }

        const user = result.rows[0]

        const validPassword =
            await bcrypt.compare(
                password,
                user.password
            )

        if(!validPassword) {

            return res.status(401).json({

                error: "Invalid password"
            })
        }

        const token = jwt.sign(

            {
                id: user.id,
                email: user.email
            },

            "secret_key",

            {
                expiresIn: "7d"
            }
        )

        res.json({

    token,

    user: {

        id: user.id,

        email: user.email
    }
})

    } catch(err) {

        console.log(err.message)

        res.status(500).json({
```

```
            error: "Server error"
        })
    }
}

module.exports = {
    register,
    login
}
```

Лістинг А.5 - Код маршруту додавання та отримання автомобіля

```
const express = require("express")

const router = express.Router()

const {
    addVehicle,
    getVehicleByUser
} = require("../controllers/vehicleController")

router.post("/add", addVehicle)

router.get("/:user_id", getVehicleByUser)

module.exports = router
```

Лістинг А.6 - Код контролера роботи з автомобілями

```
const pool = require("../database/db")

const addVehicle = async (req, res) => {

    try {

        const {
            user_id,
            vin,
            brand,
            model,
            year
        } = req.body

        const result = await pool.query(

            `
            INSERT INTO vehicles
            (
                user_id,
                vin,
                brand,
                model,
```

```
                year
            )

            VALUES ($1, $2, $3, $4, $5)

            RETURNING *
            `,

            [
                user_id,
                vin,
                brand,
                model,
                year
            ]
        )

        res.status(201).json({

            message: "Vehicle added",

            vehicle: result.rows[0]
        })

    } catch(err) {

        console.log(err.message)

        res.status(500).json({

            error: "Server error"
        })
    }
}

const getVehicleByUser = async (req, res) => {

    try {

        const { user_id } = req.params

        const result = await pool.query(

            `
            SELECT * FROM vehicles
            WHERE user_id = $1
            ORDER BY id DESC
            LIMIT 1
            `,

            [user_id]
        )
```

```
        res.json(result.rows[0])

    } catch(err) {

        console.log(err.message)

        res.status(500).json({

            error: "Server error"
        })
    }
}

module.exports = {
    addVehicle,
    getVehicleByUser
}
```

Лістинг А.7 - Код маршруту отримання телеметричних даних

```
const express = require("express")

const router = express.Router()

const {
    addTelemetry,
    getTelemetry
} = require("../controllers/telemetryController")

router.post("/add", addTelemetry)

router.get("/:vehicle_id", getTelemetry)

module.exports = router
```

Лістинг А.8 - Код контролера телеметричних даних

```
const pool = require("../database/db")
const bcrypt = require("bcryptjs")
const jwt = require("jsonwebtoken")

const register = async (req, res) => {

    try {

        const { email, password } = req.body

        const hashedPassword =
            await bcrypt.hash(password, 10)

        const result = await pool.query(
```

```
            `
            INSERT INTO users (email, password)
            VALUES ($1, $2)
            RETURNING id, email
            `,
            [email, hashedPassword]
        )
        res.status(201).json({
            message: "User created",
            user: result.rows[0]
        })
    } catch (err) {
        console.log(err.message)
        res.status(500).json({
            error: "Server error"
        })
    }
}
const login = async (req, res) => {
    try {
        const { email, password } = req.body
        const result = await pool.query(
            `
            SELECT * FROM users
            WHERE email = $1
            `,
            [email]
        )
        if(result.rows.length === 0) {
            return res.status(401).json({
                error: "Invalid email"
            })
        }
```

```
        const user = result.rows[0]

        const validPassword =
            await bcrypt.compare(
                password,
                user.password
            )

        if(!validPassword) {

            return res.status(401).json({

                error: "Invalid password"
            })
        }

        const token = jwt.sign(

            {
                id: user.id,
                email: user.email
            },

            "secret_key",

            {
                expiresIn: "7d"
            }
        )

        res.json({

    token,

    user: {

        id: user.id,

        email: user.email
    }
})

    } catch(err) {

        console.log(err.message)

        res.status(500).json({

            error: "Server error"
        })
    }
}
```

```
module.exports = {
    register,
    login
}
```

Лістинг А.9 - Код інтерфейсу API для роботи з автомобілями

```
package com.example.vehiclemsystem.data.remote.api

import com.example.vehiclemsystem.data.remote.dto.VehicleDto
import retrofit2.Response
import retrofit2.http.GET
import retrofit2.http.POST
import retrofit2.http.Body
import retrofit2.http.Path

interface VehicleApi {

    @POST("api/vehicles/add")
    suspend fun addVehicle(

        @Body dto: VehicleDto

    ): Response<VehicleDto>

    @GET("api/vehicles/{user_id}")
    suspend fun getVehicle(

        @Path("user_id")
        userId: Int

    ): Response<VehicleDto>
}
```

Лістинг А.10 - Код репозиторію роботи з автомобілями

```
package com.example.vehiclemsystem.data.repository

import com.example.vehiclemsystem.data.remote.api.RetrofitClient
import com.example.vehiclemsystem.data.remote.dto.VehicleDto

class VehicleRepository {

    suspend fun addVehicle(
        dto: VehicleDto
    ): VehicleDto? {

        return try {

            val response =
                RetrofitClient
                    .vehicleApi
```

```
                    .addVehicle(dto)

            response.body()

        } catch(err: Exception) {

            null
        }
    }

    suspend fun getVehicle(
        userId: Int
    ): VehicleDto? {

        return try {

            val response =
                RetrofitClient
                    .vehicleApi
                    .getVehicle(userId)

            response.body()

        } catch(err: Exception) {

            null
        }
    }
}
```

Лістинг А.11 - Код екрану додавання автомобіля

```
package com.example.vehiclemsystem.ui.vehicle

import androidx.compose.foundation.background
import androidx.compose.foundation.border
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material3.Button
import androidx.compose.material3.ButtonDefaults
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.OutlinedTextFieldDefaults
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
```

```
import androidx.navigation.NavController
import com.example.vehiclemsystem.navigation.Routes
import com.example.vehiclemsystem.ui.theme.AccentYellow
import com.example.vehiclemsystem.ui.theme.BackgroundDark
import com.example.vehiclemsystem.ui.theme.CardDark
import androidx.compose.ui.platform.LocalContext
import com.example.vehiclemsystem.data.local.VehicleStorage
import com.example.vehiclemsystem.data.local.SessionStorage
import com.example.vehiclemsystem.data.remote.dto.VehicleDto
import com.example.vehiclemsystem.data.repository.VehicleRepository
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

@Composable
fun AddVehicleScreen(navController: NavController) {

    val vin = remember {
        mutableStateOf("")
    }

    val brand = remember {
        mutableStateOf("")
    }

    val model = remember {
        mutableStateOf("")
    }

    val year = remember {
        mutableStateOf("")
    }
    val context = LocalContext.current

    val storage =
        VehicleStorage(context)
    val session =
        SessionStorage(context)

    val repository =
        VehicleRepository()

    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(BackgroundDark)
            .padding(24.dp),

        verticalArrangement = Arrangement.spacedBy(16.dp)
    ) {

        Text(
            text = "Add Vehicle",
```

```
        style = MaterialTheme.typography.headlineMedium,
        color = AccentYellow
    )

    OutlinedTextField(
        value = vin.value,

        onValueChange = {
            vin.value = it
        },

        modifier = Modifier.fillMaxWidth(),

        label = {
            Text("VIN")
        },

        colors = OutlinedTextFieldDefaults.colors(
            focusedBorderColor = AccentYellow,
            focusedTextColor = AccentYellow,
            unfocusedTextColor = AccentYellow
        )
    )

    OutlinedTextField(
        value = brand.value,

        onValueChange = {
            brand.value = it
        },

        modifier = Modifier.fillMaxWidth(),

        label = {
            Text("Brand")
        },

        colors = OutlinedTextFieldDefaults.colors(
            focusedBorderColor = AccentYellow,
            focusedTextColor = AccentYellow,
            unfocusedTextColor = AccentYellow
        )
    )

    OutlinedTextField(
        value = model.value,

        onValueChange = {
            model.value = it
        },

        modifier = Modifier.fillMaxWidth(),
```

```
        label = {
            Text("Model")
        },

        colors = OutlinedTextFieldDefaults.colors(
            focusedBorderColor = AccentYellow,
            focusedTextColor = AccentYellow,
            unfocusedTextColor = AccentYellow
        )
    )

    OutlinedTextField(
        value = year.value,

        onValueChange = {
            year.value = it
        },

        modifier = Modifier.fillMaxWidth(),

        label = {
            Text("Year")
        },

        colors = OutlinedTextFieldDefaults.colors(
            focusedBorderColor = AccentYellow,
            focusedTextColor = AccentYellow,
            unfocusedTextColor = AccentYellow
        )
    )

    Button(

        onClick = {

            vin.value = "1G6D25RK9N0123456"
            brand.value = "Cadillac"
            model.value = "CT5-V"
            year.value = "2022"
        },

        modifier = Modifier
            .fillMaxWidth()
            .border(
                width = 1.dp,
                color = AccentYellow,
                shape = RoundedCornerShape(20.dp)
            ),

        shape = RoundedCornerShape(20.dp),

        colors = ButtonDefaults.buttonColors(
            containerColor = CardDark
```

```
            )
        ) {

            Text(
                text = "SCAN VIA OBD-II",
                color = AccentYellow
            )
        }

        Button(

            onClick = {
                storage.saveVehicle(
                    vin = vin.value,
                    brand = brand.value,
                    model = model.value,
                    year = year.value
                )
                CoroutineScope(
                    Dispatchers.IO
                ).launch {
                    println("USER ID = ${session.getUserId()}")
                    println("VEHICLE = ${vin.value} ${brand.value}
${model.value}")
                    repository.addVehicle(

                        VehicleDto(

                            user_id =
                            session.getUserId(),

                            vin = vin.value,

                            brand = brand.value,

                            model = model.value,

                            year = year.value.toIntOrNull() ?: 2022
                        )
                    )
                }
                navController.navigate(
                    Routes.Dashboard.route
                )
            },

            modifier = Modifier
                .fillMaxWidth()
                .border(
                    width = 1.dp,
                    color = AccentYellow,
                    shape = RoundedCornerShape(20.dp)
                ),
```

```
                shape = RoundedCornerShape(20.dp),

                colors = ButtonDefaults.buttonColors(
                    containerColor = CardDark
                )
            ) {

                Text(
                    text = "SAVE VEHICLE",
                    color = AccentYellow
                )
            }
        }
}
```

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ

Рисунок Б.1 - Слайд 1

Рисунок Б.2 - Слайд 2

АКТУАЛЬНІСТЬ

– Зростання кількості сучасних автомобілів із електронними системами керування потребує постійного контролю технічного стану транспортного засобу
– Система моніторингу дозволяє контролювати основні параметри автомобіля: швидкість, оберти двигуна, температуру двигуна та напругу акумулятора
– Використання клієнт-серверної архітектури забезпечує збереження інформації користувачів та телеметричних даних у базі даних PostgreSQL

Рисунок Б.3 - Слайд 3

ДЖЕРЕЛО ДАНИХ

– ECU автомобіля формує телеметричні параметри
– Доступ до даних здійснюється через роз'єм OBD-II
– Адаптер ELM327 виконує зчитування PID-параметрів
– Передача даних до мобільного застосунку здійснюється через Bluetooth
– Отримані дані передаються на сервер та зберігаються у PostgreSQL

Рисунок Б.4 - Слайд 4

Рисунок Б.5 - Слайд 5

Рисунок Б.6 - Слайд 6

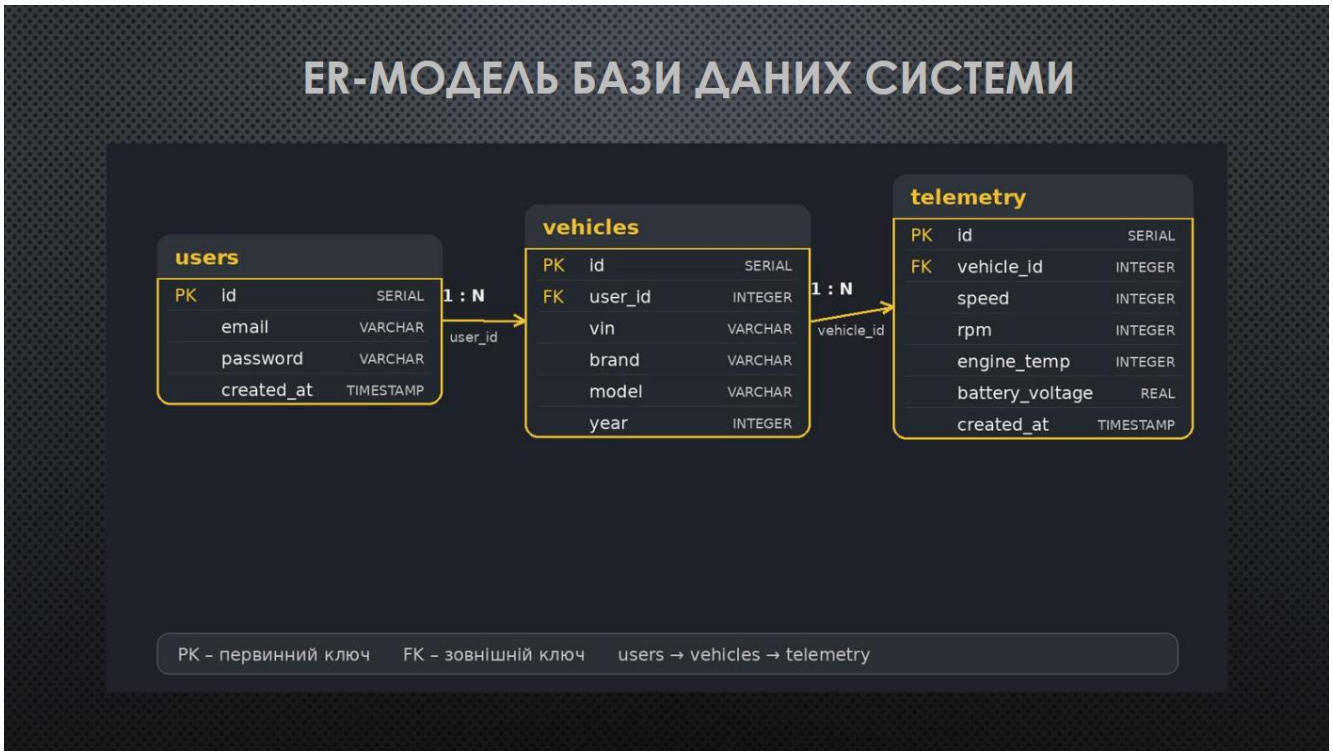

Рисунок Б.7 - Слайд 7

Рисунок Б.8 - Слайд 8

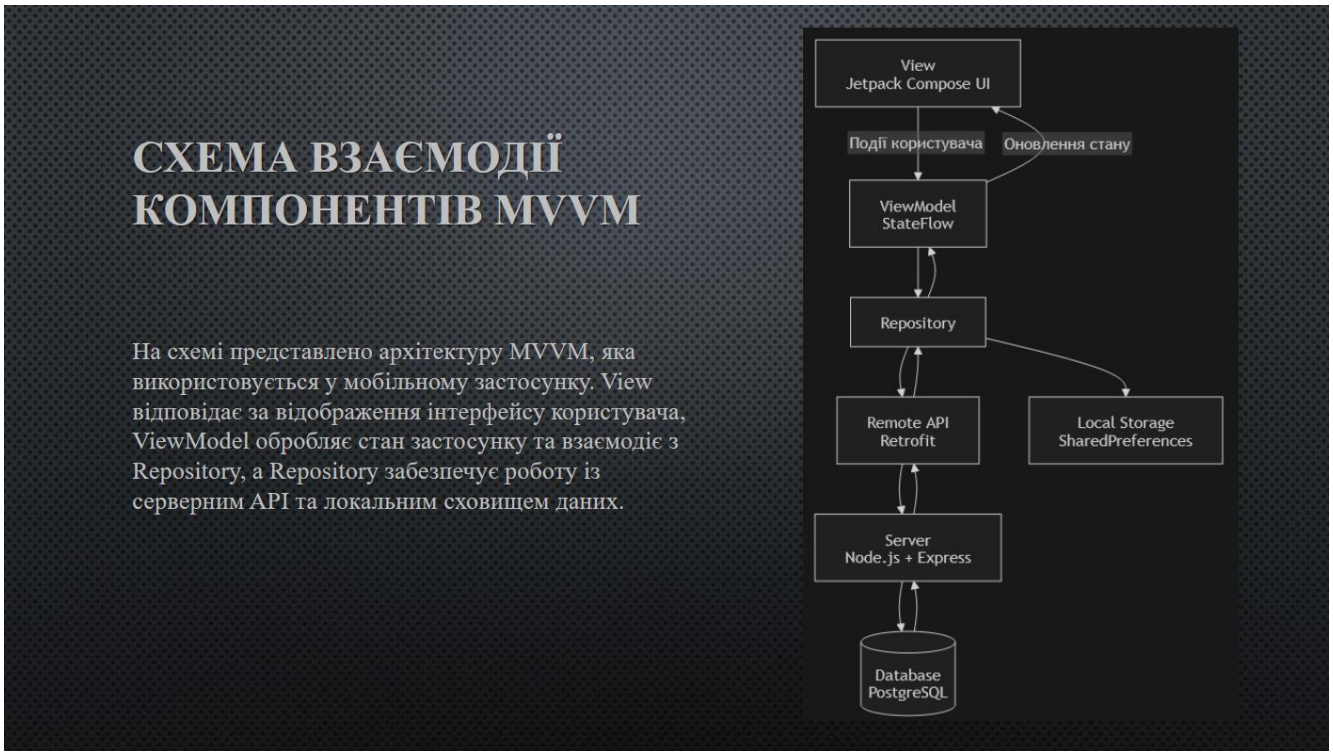

Рисунок Б.9 - Слайд 9

Рисунок Б.10 - Слайд 10

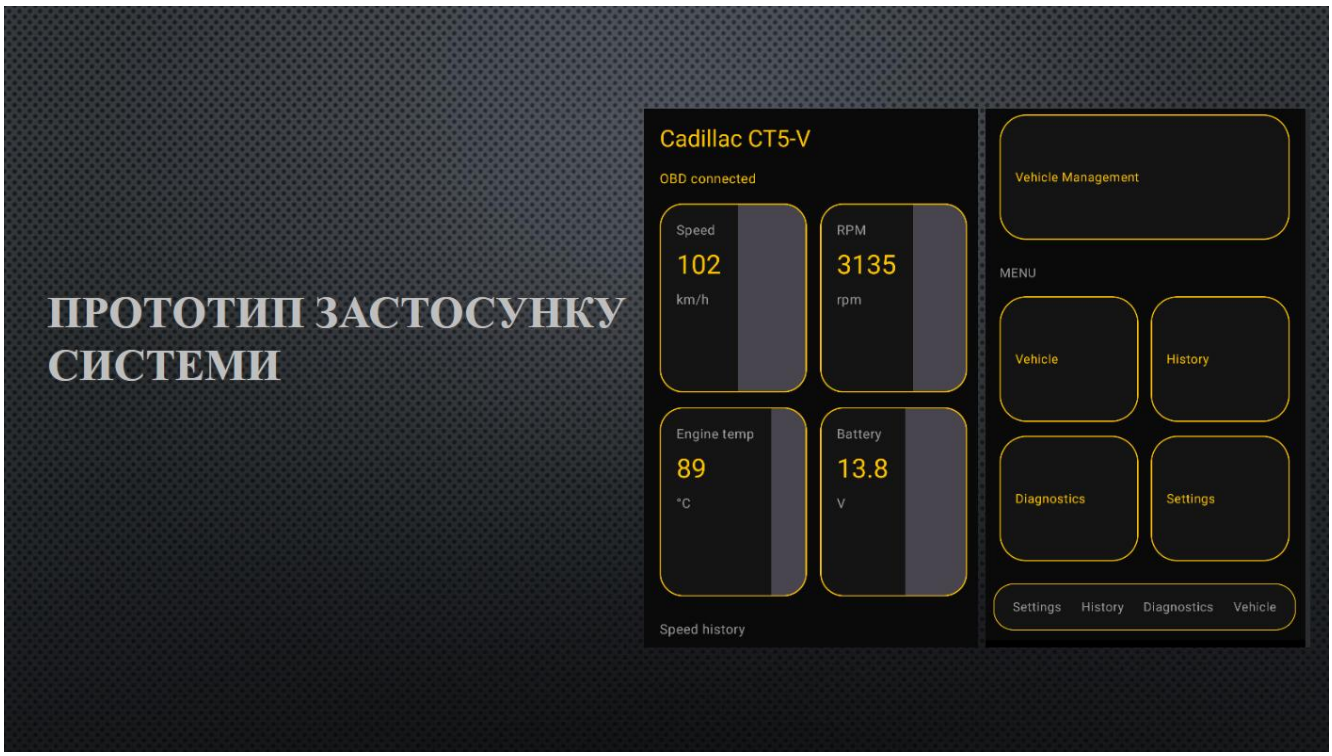

Рисунок Б.11 - Слайд 11

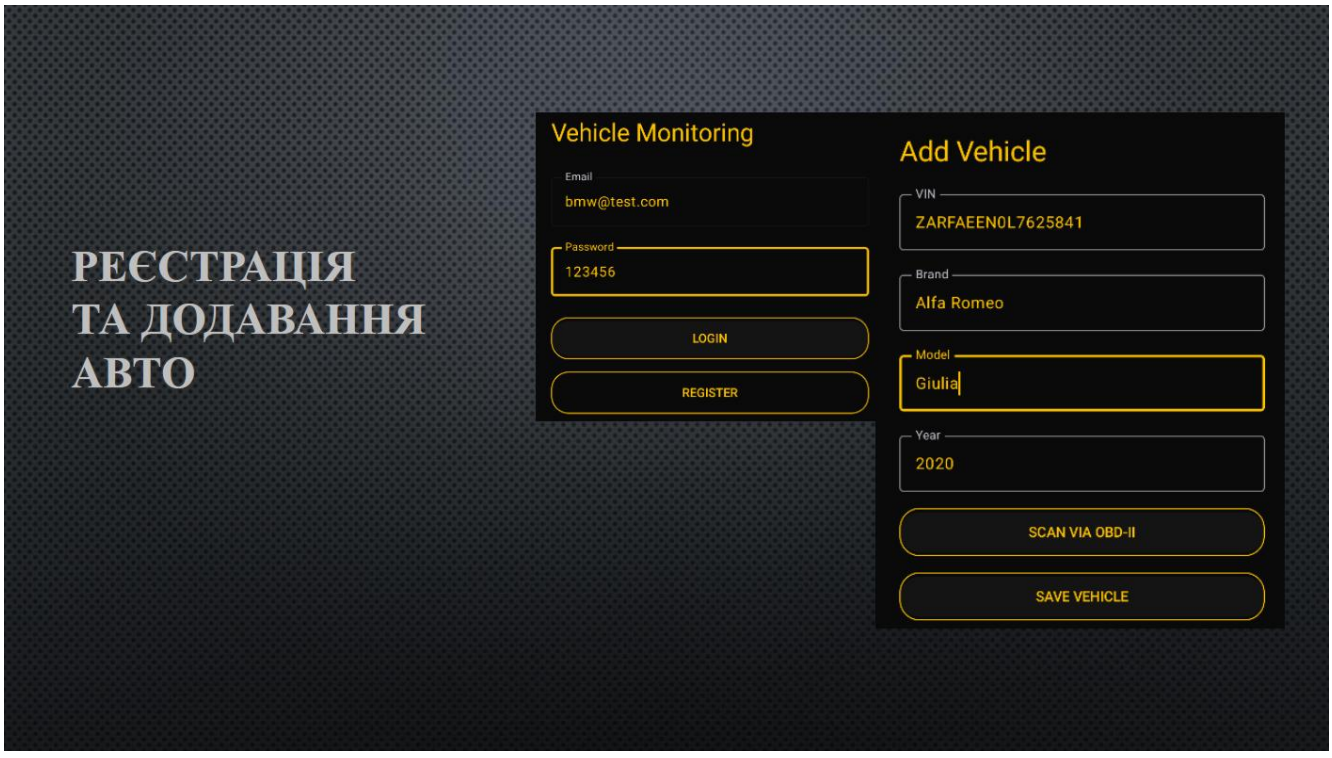

Рисунок Б.12 - Слайд 12

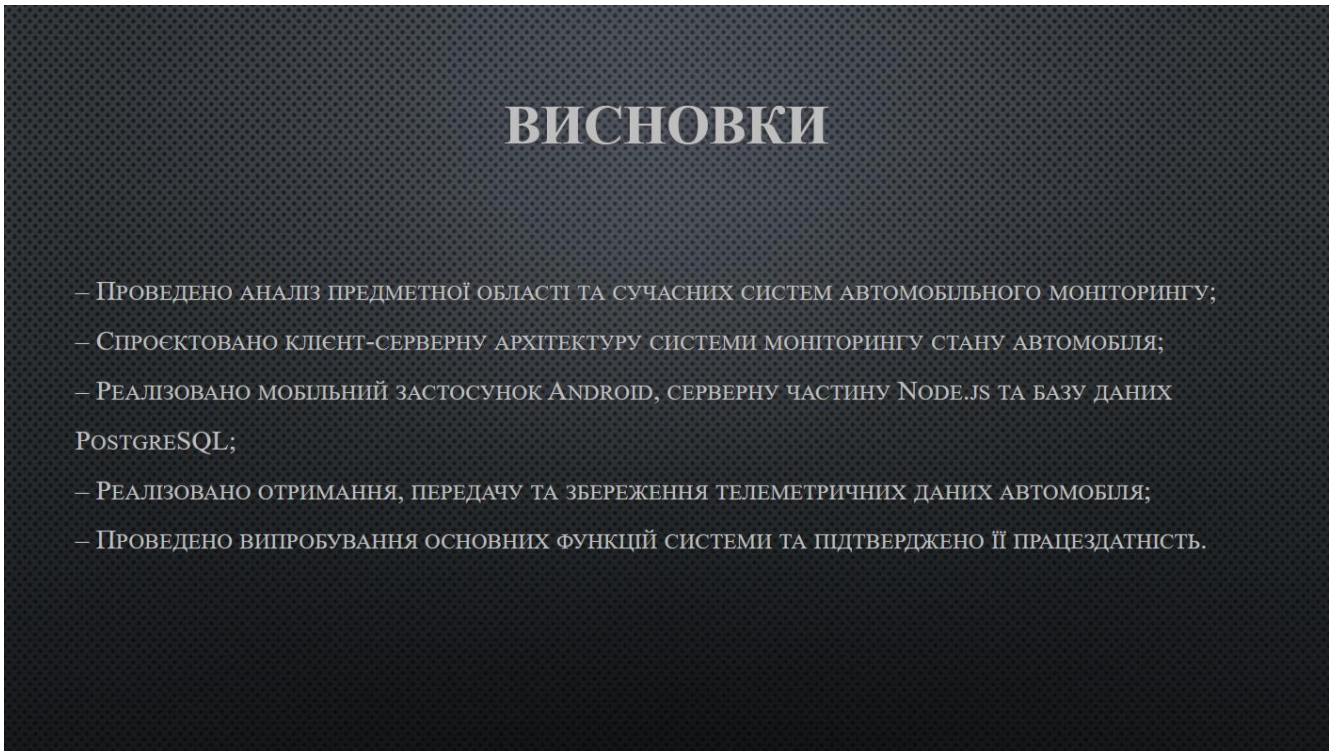

Рисунок Б.13 - Слайд 13