Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Мобільний застосунок для управління логістикою малих підприємств»

Виконав: студент 2 курсу, групи 1КІ-23мс
спеціальності 123 — «Комп'ютерна інженерія»
освітня програма – «Комп'ютерна інженерія»
Гринчак В.І.
Керівник: к.т.н., доц. каф. ОТ
Кадук О. В.
Рецензент: д.т.н., проф., голова секції УБ каф.МБІС
Яремчук Ю.Є.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

«13» 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо—кваліфікаційний рівень бакалвр
Галузь знань — 12 Інформаційні технології
Спеціальність — 123 Комп'ютерна інженерія
Освітня програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н.проф._____Азаров О. Д.
«21» березня 2025 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гринчаку Василю Івановичу

1 Тема роботи: «Мобільний застосунок для управління логістикою малих підприємству», керівник роботи Кадук О.В., доцент кафедри ОТ затверджені наказом вищого навчального закладу від «20» березня 2025 року № 97

2 Термін подання студентом роботи 13.05.2025

3 Вихідні дані до роботи: координати водія; особисті дані користувача; дані про маршрут; сучасні технології розробки мобільних застосунків.

4 Зміст розрахунково-пояснювальної записки Вступ. Техніко-економічне обґрунтування доцільності розробки мобільного застосунку для управління логістикою малих підприємств. Розробка архітектури та функціональних рішень. Реалізація програмного забезпечення. Тестування та впровадження мобільного застосунку. Висновки.

5 Перелік графічного матеріалу Схема структури зберігання даних у хмарі. Алгоритм дій модуля MainActivity. Дизайн інтерфейсу activity_main.xml. Алгоритм дій модуля NavigationFragment. Дизайн інтерфейсу fragment_navigation.xml. Вигляд графічного інтерфейсу мобільного застосунку.

6 Консультанти розділів роботи приведені в таблиці 1

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 - 4	доцент кафедри ОТ Кадук О.В.	21.03.25	13.05.25
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.25	30.05.25

7 Дата видачі завдання 21.03.2025

8 Календарний план приведений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва та зміст етапу	Строк виконання	Підпис
1.	Постановка задачі роботи	21.03.25	
2.	Пошук матеріалів та інформаційних джерел	21.03.25—30.03.25	
3.	Аналіз сучасного стану та проблем у сфері логістики малих підприємств	21.03.25—30.03.25	
4.	Формулювання вимог до мобільного застосунку	31.03.25—13.04.25	
5.	Програмна реалізація мобільного застосунку	14.04.25—27.04.25	
6.	Тестування мобільного застосунка на різних пристроях	21.04.25—27.04.25	
7.	Оформлення пояснювальної записки та графічної частини	28.04.25—11.05.25	
8.	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.2025	

Студент Василь ГРИНЧАК

Керівник роботи к.т.н., доц. каф. ОТ Олександр КАДУК

АНОТАЦІЯ

УДК 004.42

Гринчак В. І. Мобільний застосунок для управління логістикою малих підприємств. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Комп'ютерна інженерія». Вінниця: ВНТУ, 2025. 81 с.

На укр. мові. Бібліогр.: 23 назв; рис.: 22; табл. 7.

У бакалаврській кваліфікаційній роботі розроблено мобільний застосунок для управління логістикою малих підприємств. Застосунок включає реєстрацію користувачів із розмежуванням ролей, створення маршрутів постачальником, передачу маршруту водієві, відображення карти з точками, автоматичне оновлення координат у реальному часі та завершення маршруту. У загальній частині роботи розглянуто особливості логістики малих підприємств. У технологічній частині виконана розробка програмного забезпечення, з використанням хмарний технологій, а саме Firebase. У частині тестування, виконано тестування системи на різних пристроях.

Ключові слова: логістика, Firebase, Google Maps API, Android, Java.

ABSTRACT

Grinchak V. I. Mobile application for logistics management of small enterprises. Bachelor's qualification work in speciality 123 ‘Computer Engineering’, educational programme ‘Computer Engineering’. Vinnytsia: VNTU, 2025. 81 p.

In Ukrainian language. Bibliographer: 23 titles; Fig.: 22; table 7.

In the bachelor's qualification work, a mobile application for logistics management for small businesses developed. The application includes user registration with role delineation, route creation by the supplier, route transfer to the driver, map display with points, automatic real-time coordinate updates, and route completion. The general part of the paper discusses the peculiarities of logistics for small enterprises. In the technological part, the software was developed using cloud technologies, namely Firebase. In the testing part, the system was tested on various devices.

Keywords: logistics, Firebase, Google Maps API, Android, Java.

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації бізнесу логістика стає одним із найважливіших елементів ефективного функціонування підприємств. Особливо це питання стоїть для малих підприємствам, які, не мають доступу до комплексних IT-рішень для автоматизації процесів. Багато подібних підприємств досі використовують ручні засоби управління доставкою, наприклад таблиці Excel, телефонні дзвінки та месенджери. Подібні дії призводить до неузгодженості дій, втрати часу, зростання витрат і зниження якості обслуговування.

Останніми роками питання оптимізації логістичних процесів активно досліджуються як у технічній, так і в економічній науці. Зокрема, велика увага приділяється інтеграції мобільних технологій із хмарними сервісами, такими як Firebase, Google Maps, а також аналізу ефективності доставки в режимі реального часу. Проте більшість таких рішень орієнтовані на середній і великий бізнес та є недоступними для впровадження малими підприємствами через високу вартість або складність використання.

Метою дослідження є створення мобільного застосунку для управління логістикою малого підприємства з можливістю реєстрації нових користувачів, формування маршрутів доставки, відстеження водіїв у реальному часі та збереження історії поїздок.

Об’єктом дослідження є процес логістичної доставки на малому підприємстві, а саме інформаційна взаємодія між постачальником та водієм.

Предметом дослідження є методи та інструменти розробки мобільного застосунку для управління логістичними маршрутами із використанням хмарних технологій.

Задачі, які необхідно виконати для досягнення поставленої мети:

— провести аналіз особливості організації логістики в малих підприємствах та визначити основні проблеми, що виникають у процесі координації доставок;

— ознайомитися з існуючими програмними рішеннями та оцінити їхню придатність для використання у малому бізнесі;

— розробити архітектуру мобільного застосунку з урахуванням взаємодії з хмарними сервісами;

— провести функціональне тестування системи на реальному пристрої та в середовищі емулятора, оцінити стабільність, продуктивність і готовність до впровадження.

Апробацією результатів кваліфікаційної роботи є подання тези на Молодь в науці: дослідження, проблеми, перспективи (МН-2025).

Створено **публікацію** Мобільний застосунок для управління логістикою малих підприємств // Гринчак В. І., Кадук О. В. Матеріали Молодь в науці: дослідження, проблеми, перспективи (МН-2025) (Вінниця, 2025 р.) [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/author/submission/24937

1 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ЛОГІСТИКОЮ МАЛИХ ПІДПРИЄМСТВ

1.1 Аналіз сучасного стану та проблем у сфері логістики малих підприємств

На сьогоднішній час, багато малих підприємств мають потребу у перевезенні вантажів різного типу. Через високу вартість спеціалізованих комплексних рішень, більшість з них віддає перевагу використанню загальнодоступних, переважно безкоштовних застосунків. Для таких підприємств логістичні процеси часто базуються на телефонних дзвінках, Excel-таблицях або паперових журналах. Унаслідок цього виникає затримка у виконанні замовлень, плутанина у черговості точок доставки, а також труднощі в оцінці ефективності роботи водіїв.

Водночас ринок мобільних рішень стрімко розвивається, і багато галузей впроваджують цифрові інструменти для автоматизації. Проте саме малий бізнес часто опиняється поза цим прогресом через обмежені ресурси, відсутність технічних знань або готових адаптованих продуктів за доступною ціною.

Розроблений застосунок має вирішити цю проблему, дозволяючи підприємствам оперативно створювати маршрути з урахуванням проміжних зупинок, супроводжувати працівника в реальному часі та надавати керівнику змогу відслідковувати пересування перевізника.

Метою проекту є підтримка малих підприємств у сфері логістики шляхом оптимізації процесів планування маршрутів та контролю перевезень, що сприятиме зниженню витрат, пов’язаних із транспортуванням.

1.2 Аналіз існуючих аналогів та їх порівняння

Існує чимало рішень для управління логістикою, проте більшість із них орієнтовані на великі компанії або потребують платної підписки. Розглянемо переваги та недоліки популярних сервесів.

Сервіс Google Maps є безкоштовним картографічним інструментом, створений компанією Google, що включає не лише інтерактивну мапу, а й набір функціональних можливостей, побудованих на основі сучасних геоінформаційних

технологій. Це один із найпопулярніших сервісів у світі для навігації та прокладання маршрутів, який підтримує роботу на різних платформах, має інтуїтивно зрозумілий графічний інтерфейс і високий рівень інтеграції з сервісами Google [1].

Google Maps також надає користувачам доступ до великої кількості корисної інформації, зокрема даних про громадський транспорт, інтенсивність руху в реальному часі, можливість перегляду вулиць завдяки режиму Street View, а також деталізовану інформацію про об’єкти інфраструктури, наприклад заклади, установи, будівлі тощо. Завдяки цим функціям сервіс широко використовується як для особистих, так і для бізнес-потреб.

Незважаючи на свою популярність і функціональність, Google Maps має певні обмеження. Наприклад, він не завжди ефективно виконує завдання з побудови оптимального маршруту, якщо йдеться про обробку декількох точок доставки або відвідування в межах одного маршруту. Такий недолік є суттєвим для логістичних задач малих підприємств, де важливо мінімізувати витрати часу і пального. Графічний інтерфейс Google maps представлений на рисунку 1.1 [2].

OpenStreetMap, далі OSM, це відкритий картографічний проект, який надає вільний доступ до географічних даних усього світу. Проект створений на основі принципів краудсорсингу, тобто, тисячі користувачів по всьому світу наповнюють карту, редагують її та актуалізують дані. Відкритість і доступність даних роблять OSM привабливою альтернативою до комерційних рішень, особливо для розробників спеціалізованих застосунків [3].

Перевагами OSM у побудові маршрутів є відкритий доступ до даних, що дозволяє вільно використовувати карти та інформацію для розробки власних сервісів без ліцензійних обмежень. Розробники мають можливість повністю контролювати логіку побудови маршрутів, зокрема з урахуванням кількох проміжних точок, типу транспорту, обмежень або особливих умов. Також у багатьох регіонах, особливо менш охоплених комерційними сервісами, OSM надає більш точну та актуальну інформацію завдяки активності місцевих спільнот.

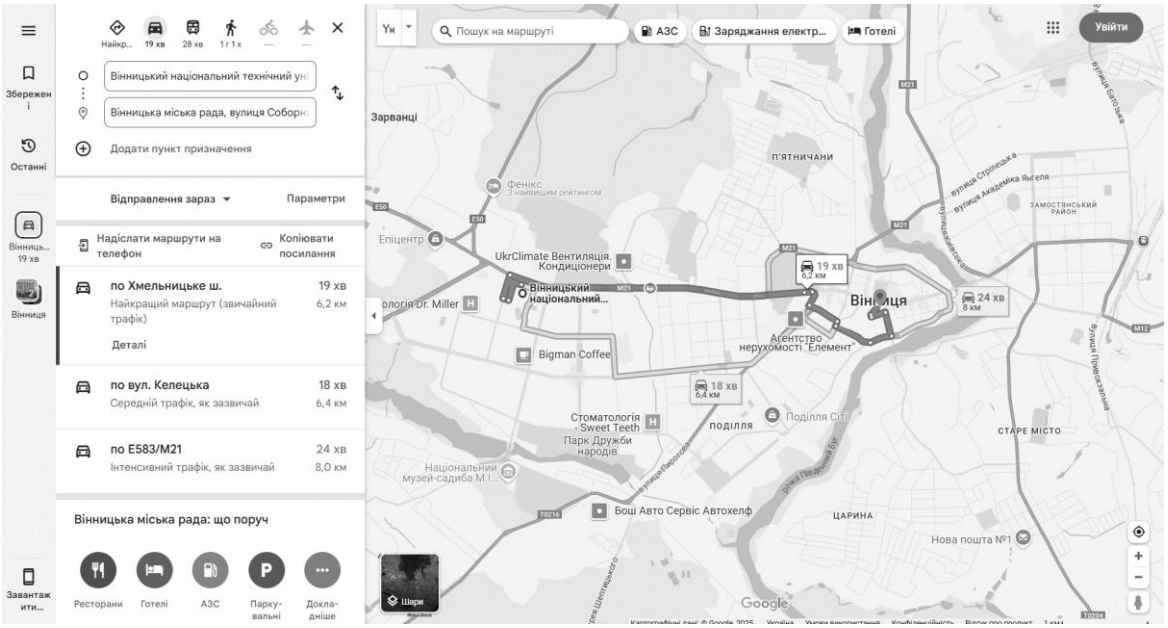

Рисунок 1.1 — Графічний інтерфейс Google maps

Проте недоліком OSM у порівнянні з комерційними сервісами є те, що платформа не має вбудованої функції маршрутизації, як у Google Maps. На основі її відкритих даних існує багато сторонніх інструментів і API, які реалізують побудову маршрутів, такі рішення не завжди є зручними для використання.

Нерівномірна якість даних може повипливати на точність і повноту карти, тому що це залежить від активності користувачів у конкретному регіоні. У деяких сільських або малозаселених зонах дані можуть бути неповними або застарілими.

OSM не має єдиного універсального застосунку з широким функціоналом для побудови маршрутів. Більшість рішень потрібно створювати вручну або інтегрувати сторонні сервіси. Графічний інтерфейс OpenStreetMap представлений на рисунку 1.2.

OptimoRoute – це комерційний хмарний сервіс, призначений для оптимізації маршрутів доставки, обслуговування клієнтів і виїзних робіт. Цей сервіс є одним із найпопулярніших рішень у сфері логістики, що активно використовується підприємствами малого та середнього бізнесу в США, ЄС та інших країнах [4].

Потужна оптимізація маршрутів є великою перевагою сервісу. Він автоматично розраховує найоптимальніші маршрути з урахуванням численних точок доставки, часових вікон, пріоритетів, обмежень транспортних засобів тощо.

Рисунок 1.2 — Графічний інтерфейс OpenStreetMap

Сервіс надає підтримку в реальному часі, а саме система відслідковує місцезнаходження водіїв і дозволяє диспетчеру бачити прогрес виконання маршрутів онлайн.

Також, система включає функцію аналітики та створення звітності, доступ до розширених звітів про витрати часу, пробіг, ефективність логістики, продуктивність працівників.

OptimoRoute підтримує інтеграцію з Google Sheets, CRM-системами, API для автоматизації та зручної роботи вже з наявними даними.

Проте основним недоліком OptimoRoute є недоступність для України. Станом на сьогодні сервіс не підтримує побудову маршрутів по території України, оскільки в його географічній базі маршрутизація для нашої країни не активована.

Сервіс є платним, але є пробна безкоштовна версія, яка має суттєві обмеження по кількості маршрутів, точок і користувачів.

OptimoRoute не дозволяє змінювати або адаптувати маршрутизатор під специфічні потреби, на відміну від рішень на базі OpenStreetMap.

Також недоліком є залежність від інтернет-з'єднання. Для роботи потрібен постійний доступ до мережі, оскільки всі розрахунки виконуються на сервері компанії. Графічний інтерфейс OptimoRoute представлений на рисунку 1.3.

Рисунок 1.3 — Графічний інтерфейс OptimoRoute

Більшість із зазначених аналогів не вирішують одразу всі задачі малого підприємства: зручне створення маршрутів, контроль виконання завдань, зв'язок із водієм у реальному часі та простий інтерфейс. У зв'язку з цим виникає потреба у створенні власного мобільного застосунку, який повністю відповідатиме реальним процесам та потребам підприємства.

1.3 Обґрунтування необхідності розробки власного рішення

Метою створення мобільного застосунку є забезпечення малих підприємств

ефективним інструментом для управління логістикою, який дозволить:

— спростити планування перевезень;

— контролювати виконання маршрутів у режимі реального часу;

— зменшити кількість помилок у комунікації;

— оптимізувати витрати на доставку.

Для досягнення поставлених цілей необхідно розв’язати такі основні задачі:

— реалізувати реєстрацію користувачів у системі та визначити роль користувача: водій, постачальник;

— забезпечити швидку побудову оптимального маршруту із використанням Google Maps API;

— реалізувати механізм оновлення геолокації водія і відображення його на мапі в реальному часі;

— надати постачальнику можливість переглядати активні маршрути;

— реалізувати збереження історії поїздок та можливість повторного використання маршрутів.

На відміну від сторонніх сервісів, власне рішення повністю адаптоване під внутрішні процеси підприємства, його можна легко доповнювати новим функціоналом, враховуючи індивідуальні бізнес-процеси.

Отже, власна розробка забезпечує гнучкість, економічну доцільність і краще задоволення потреб кінцевих користувачів.

1.4 Формулювання вимог до мобільного застосунку

Вимоги до мобільного застосунку визначаються як із технічної, так і з функціональної точки зору, з урахуванням цілей розробки та особливостей логістичних процесів малого підприємства.

Технічні вимоги мають бути наступним:

— платформа: Android, версія не нижче 8.0;

— наявність Google Services;

— підключення до Інтернету для синхронізації з Firebase;

— наявність GPS-модуля для відстеження геолокації;

— підтримка Google Maps API для побудови та візуалізації маршрутів;

— застосування Firebase Realtime Database або Firestore для зберігання даних маршруту, користувачів та статусів поїздок [5,6].

Функціональні вимоги мають бути наступні:

— система автентифікації та авторизації користувачів;

— створення маршруту з множинними контрольними точками через візуальний редактор на мапі;

— автоматичний розрахунок найкоротшого шляху між точками доставки;

— відображення поточного місцезнаходження водія на мапі в реальному часі з інтервалом оновлення;

— можливість водієм позначити точку як "пройдену", з фіксацією часу проходження;

— збереження історії поїздок із можливістю фільтрації, перегляду деталей і повторного використання маршруту;

— автоматичне очищення старих або завершених маршрутів для оптимізації зберігання.

Інтерфейсні вимоги:

— мінімалістичний, зрозумілий інтерфейс, з розділенням за ролями користувачів;

— підтримка української мови, чітка навігація між основними екранами;

— інтуїтивно зрозуміле позначення маршрутних точок, активного маршруту, пройдених ділянок;

— можливість швидкої навігації водієм до наступної точки маршруту через вбудовану навігацію Google Maps.

Вимоги безпеки до проекту:

— авторизований доступ до даних маршруту;

— шифрування даних при передачі через Firebase;

— захист від несанкціонованого редагування активних маршрутів.

Отже, створені вимоги дозволяють забезпечити стабільність роботи застосунку, відповідність поставленим цілям та високу якість користувацького досвіду.

2 РОЗРОБКА АРХІТЕКТУРИ ТА ФУНКЦІОНАЛЬНИХ РІШЕНЬ

2.1 Вибір технологій та середовища розробки

Для розробки мобільного застосування обрано мову програмування Java, далі розглянемо переваги та недоліки мов програмування, які використовуються в мобільній розробці, і порівняємо у таблиці 2.1 [7].

Таблиця 2.1 Порівняння мов програмування Java, Kotlin та Xamarin.

Java(android)	**Kotlin**	**Xamarin(C#)**
Є об’єктно-орієнтованою мовою програмування, що дозволяє створювати модульні застосунки з можливістю повторного використання коду [8]	Також є об'єктно орієнтованою мовою [9]	Також є об'єктно орієнтованою мовою, офіційно підтримку заверешено [10]
Відноситься до нативної розробки, що забезпечує повний доступ до ресурсів та функцій Android-платформи	Також належить до нативного типу розробки	Має обмежений доступ до вихідного коду платформи
Java — це перша мова, використана для розробки під Android, тому вона має велику й активну спільноту розробників	Kotlin — порівняно нова мова, тому її спільнота ще формується і є менш чисельною	Xamarin має обмежене поширення серед Android-розробників, що зумовлює невелику спільноту.
Для розробки достатньо володіти лише нативними інструментами Android	Також потребує знання лише нативних засобів розробки Android	Окрім стандартних інструментів Xamarin, необхідно додатково володіти нативними засобами для кожної цільової платформи (Android, iOS тощо)

Для реалізації зберігання даних у проєкті було обрано Firebase Realtime Database. Firebase Realtime Database – це хмарна нереляційна база даних, яка оперує даними у форматі JSON і забезпечує їх синхронізацію в реальному часі з усіма

підключеними клієнтами. Також коли створюються кросплатформні додатки Flutter та Firebase, усі клієнти можуть спільно використовувати один екземпляр бази даних Realtime та автоматично отримувати оновлення з найновішими даними [11, 12].

Продовження таблиці 2.1 Порівняння мов програмування Java, Kotlin та Xamarin

Підходить виключно для створення мобільних застосунків під операційну систему Android.	Також обмежується розробкою мобільних застосунків для Android.	Є кросплатформеним рішенням, що дозволяє створювати спільну логіку для застосунків під Android, iOS.
Розроблені застосунки зазвичай прості й компактні	За рівнем компактності та простоти програми дещо поступаються Java	Застосунки менш компактні й складніші

Вирішальними факторами для вибору цього сервісу стали його безкоштовність на базовому рівні, а також універсальність.

Серед ключових переваг Firebase Realtime Database можна виділити описане далі.

Замість класичних HTTP-запитів, база забезпечує постійне з'єднання з сервером, що дозволяє відображати будь-які зміни майже миттєво на всіх підключених пристроях. Для завдання оперативної взаємодії між диспетчером та перевізником це є оптимальним рішенням.

Навіть при втраті інтернет-з’єднання клієнт зберігає доступ до даних завдяки кешуванню в пам’яті. Після відновлення зв’язку система автоматично синхронізує усі оновлення, що відбулись під час відсутності підключення.

Завдяки зберіганню інформації локально та постійній синхронізації, запити до сервера мінімізуються, що підвищує загальну продуктивність застосунку.

Основним недоліком стало лише те, що під час розробки було незвично використовувати нереляційну базу даних, замість всім звичної реляційної бази

даних, які забезпечують більш структурований підхід до роботи з таблицями та запитами. У порівнянні з цим, гнучка, але менш структурована природа Firebase вимагає іншого підходу до організації даних.

Для реалізації мобільного застосунку було обрано платформу Android з використанням середовища розробки Android Studio.

Android Studio — це офіційне інтегроване середовище розробки для створення застосунків на платформі Android, представлене компанією Google 16 травня 2013 року під час конференції Google I/O. Графічний інтерфейс Android Studio 2024.2.2 наведено на рисунку 2.1 [13].

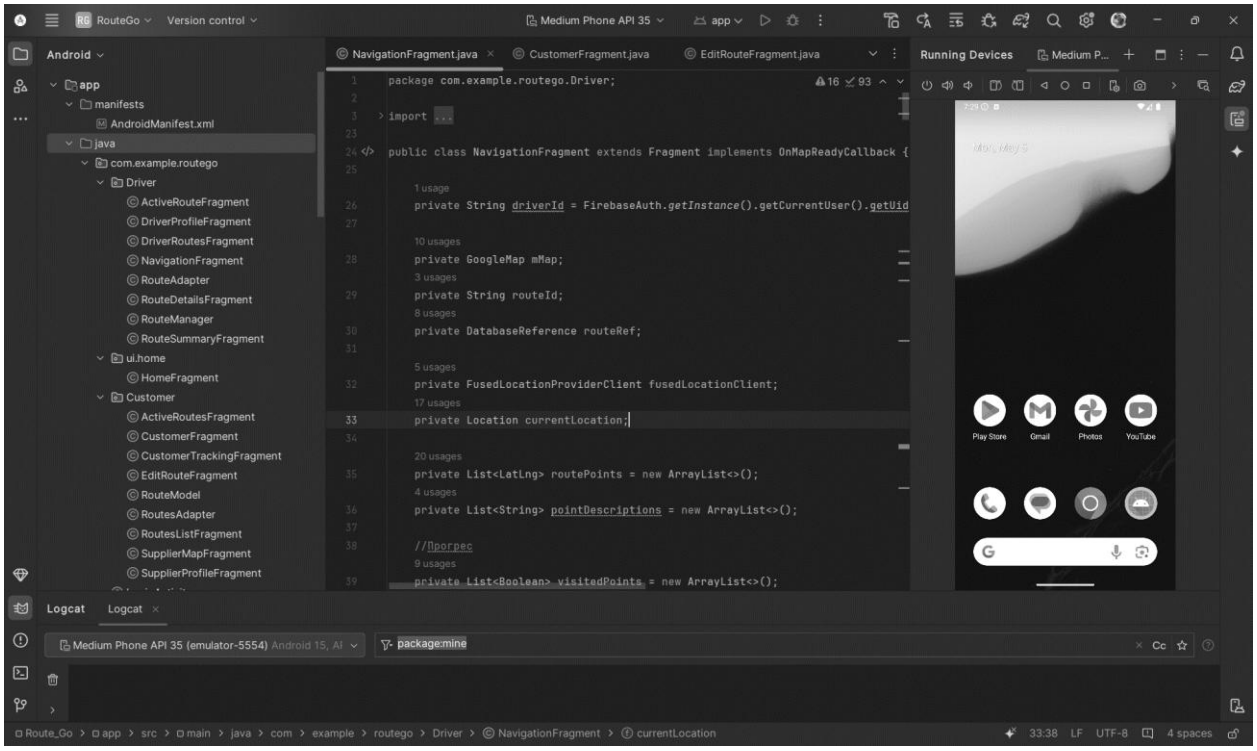

Рисунок 2.1 — Графічний інтерфейс Android Studio 2024.2.2

Це середовище прийшло на зміну попередньому рішенню — плагіну ADT для Eclipse, і було побудоване на основі відкритого коду IntelliJ IDEA Community Edition від компанії JetBrains. Android Studio розробляється у рамках відкритої моделі та розповсюджується відповідно до умов ліцензії Apache 2.0.

Середовище розробки доступне для операційних систем Windows, macOS та Linux, і забезпечує повний набір інструментів для створення застосунків не лише для смартфонів і планшетів, а й для носимих пристроїв, телевізорів, автомобільних

інформаційно-розважальних систем та розумних окулярів, Google Glas. Для розробників, які раніше використовували Eclipse із плагіном ADT, доступний інструмент для автоматичного перенесення проєктів до Android Studio.

Android Studio середовище, якеоптимізоване для виконання основних завдань розробника Android-застосунків: проєктування інтерфейсів для пристроїв із різними типами екранів, тестування сумісності з різними версіями Android, а також налагодження, складання та розгортання застосунків. На додаток до функціоналу IntelliJ IDEA, Android Studio включає нову уніфіковану систему складання, засновану на Gradle, що підтримує безперервну інтеграцію та автоматизацію процесів тестування і публікації застосунків.

Такий вибір зумовлений широким розповсюдженням Android-пристроїв, безкоштовністю інструментів розробки та можливістю інтеграції з хмарними сервісами.

Отже, обрані технології забезпечують швидку розробку, масштабованість, зручність оновлення та мінімальні витрати на обслуговування.

2.2 Розробка загальної архітектури системи

Розробка мобільного застосунку здійснювалася з урахуванням модульного підходу, що забезпечує зручність масштабування, тестування та підтримки коду.

Застосунок реалізовано за принципом Model–View–Controller. Model–View–Controller (MVC) — це архітектурний шаблон, який застосовується в процесі розробки програмного забезпечення для структурування коду [14].

Він передбачає поділ застосунку на три основні компоненти: модель , відповідає за роботу з даними, вигляд, графічний інтерфейс користувача, та контролер, обробка логіки взаємодії між моделлю та виглядом. Такий підхід дозволяє розділити логіку бізнес-процесів і візуальне представлення, що спрощує супровід коду: зміни в інтерфейсі не впливають на модель даних, і навпаки — зміни у моделі не потребують переробки інтерфейсу.

Загальна архітектура застосунку побудована за принципами розділення обов’язків (Separation of Concerns) та архітектури з використанням фрагментів.

Separation of Concerns – це один з базових принципів програмної інженерії, що передбачає розподіл відповідальностей між різними частинами програми, щоб кожна з них відповідала лише за одну конкретну функцію або аспект.

Інтерфейс користувача зосереджений у окремих фрагментах, такі як HomeFragment, DriverProfileFragment, DriverRoutesFragment, які відповідають лише за відображення інформації та прийом дій користувача.

Бізнес-логіка винесена у окремі класи, наприклад, RouteManager, що обробляє логіку побудови маршруту, переміщення, зон доставки.

Модель даних, така як User, реалізує структуру об'єктів, що зберігаються у Firebase, і передаються між фрагментами.

Такий підхід дозволяє змінювати або розширювати одну частину застосунку без впливу на інші. Наприклад, змінити візуалізацію, не змінюючи логіку маршруту або не змінюючи модель даних, яка використовується в всьому проекті.

Застосунок реалізований з використанням архітектури на основі фрагментів, що є стандартом для сучасних Android-додатків. В основі лежить центральна активність, MainActivity, яка виконує роль контейнера для фрагментів і керує навігацією між ними.

Основними перевагами використання фрагментів є перевикористання інтерфейсу, гнучка навігація та адаптивність інтерфейсу. Фрагменти можна повторно використовувати в різних частинах застосунку або навіть у кількох застосунках. Android підтримує навігаційну архітектуру на базі фрагментів, що полегшує керування стеком переходів. Також фрагменти дають змогу створювати інтерфейси, які адаптуються під різні розміри екранів.

Також реалізовано шаблон Callback, що дозволяє фрагментам отримувати повідомлення про події, наприклад, входження в зону доставки, без жорсткого зв’язування з класом, який їх генерує.

Для обробки складних сценаріїв, таких як, переміщення по маршруту або виявлення входження в зону доставки, використовується окремий клас RouteManager, який не має прямої залежності від UI, що відповідає принципу Low Coupling / High Cohesion, тобто низький зв'язок, висока згуртованість.

Таке використання принципів розділення обов’язків та фрагментної архітектури забезпечує ефективну структуру застосунку, яка легко масштабується, тестується та підтримується.

Основними модулями системи являються модуль аутентифікації, головна активність, користувацький модель, модуль водія та модуль постачальника.

Модуль аутентифікації відповідає за підключення до Firebase Realtime Database забезпечує перевірку даних користувача та створення нових облікових записів.

Головна активність є центральною точкою навігації між фрагментами додатку. Включає елементи інтерфейсу, такі як нижнє меню, та відповідає за переключення між основними розділами.

Користувацький модель – це клас моделі користувача, який містить інформацію про поточного користувача, такі дані як ім’я, роль, UID тощо. Застосовується для передачі даних між компонентами.

Модуль водія відповідає за відображення поточного маршруту, що виконується водієм, профілю водія з основною інформацією, список усіх доступних маршрутів та реалізує логіку обробки маршрутів, перегляд деталей, оновлення стану тощо.

Модуль постачальника реалізує функціональність, призначену для представників компаній або осіб, які створюють, редагують і контролюють маршрути доставки. Він охоплює як функції управління, так і відстеження у реальному часі.

Завдяки такій структурі застосунок є гнучким, розширюваним та підтримує ефективну командну роботу за рахунок чітко розділених обов’язків між модулями.

2.3 Опис бази даних та моделі даних

Оскільки використовується Firebase Realtime Database — хмарна нереляційна база даних, яка забезпечує зберігання та синхронізацію даних у форматі JSON в режимі реального часу. Це дозволяє забезпечити безперервну комунікацію між

постачальником і водієм, відображати зміни маршруту миттєво, а також підтримувати живе оновлення геопозицій [15].

JSON, скорочено від JavaScript Object Notation — це текстовий формат, призначений для обміну даними між комп'ютерами. Його основою є зрозумілий людині текст, що дозволяє описувати об'єкти та інші структури даних у зручному вигляді. Найчастіше JSON використовується для передавання структурованої інформації через мережу, зокрема завдяки процесу, який називається серіалізацією — перетворенням даних у формат, придатний для збереження чи передавання [16].

Для зручної роботи з даними виділимо основні колекції даних, які нам потрібні. Колекції представлено в таблиці 2.2.

Таблиця 2.2 — Опис колекцій даних, які зберігатимуться у хмарі.

Ім'я	**Опис**
Users	Колекція користувачів
DriversLocation	Колекція місцезнаходження водіїв
Routes	Колекція маршрутів

Розглянемо кожну колекцію даних детальніше.

Users — це колекція, яка зберігає дані про користувачів системи та їх ролі. Дані колекції Users та їх опис представлено в таблиці 2.3.

Таблиця 2.3 — Опис колекції Users

Ім'я	**Опис**
email	Електронна пошта користувача
isApproved	Чи підтверджено користувача адміністратором (true/false)
role	Роль користувача (driver або supplier)
userId	Унікальний UID користувача, використовується для входу через Firebase

Routes — це колекція, яка зберігає дані про всі маршрути та їх стан. Дані колекції Routes та їх опис представлено в таблиці 2.4.

DriversLocation — це колекція, яка зберігає дані про місцезнаходження водія, щоб постачальник міг бачити його на мапі. Дані колекції DriversLocation та їх опис представлено в таблиці 2.5.

Таблиця 2.4 — Опис колекції Routes

Ім’я	Опис
description	Опис маршруту
driverId	UID водія, який виконує маршрут
createdBy	UID постачальника, який створив маршрут
startedAt	Час початку маршруту (у форматі timestamp)
endedAt	Час завершення маршруту (у форматі timestamp)
status	Поточний статус маршруту (active, completed)
complete_seen	Чи переглянув постачальник завершення маршруту
points[]	Список точок маршруту з полями: latitude, longitude, visited, description

Таблиця 2.5 — Опис колекції DriversLocation

Ім’я	Опис
latitude	Поточна широта водія
longitude	Поточна довгота водія
timestamp	Час останнього оновлення позиції водія

Схема структури зберігання даних зображена на рисунку 2.2.

Рисунок 2.2 — Схема структури зберігання даних у хмарі

Отже, Firebase Realtime Database ідеально підходить для задач, де потрібна миттєва реакція на зміну даних, наприклад, у логістичних застосунках з живим відстеженням та оновленням маршрутів. Незважаючи на відсутність складних зв’язків як у реляційних базах, її гнучка JSON-структура, легкість інтеграції, офлайн-підтримка та низька затримка роблять її ефективним рішенням для проекту.

2.4 Опис основних функціональних можливостей застосунку

Мобільний застосунок призначений для організації та оптимізації процесу доставки вантажів з розмежуванням ролей користувачів: постачальник (supplier) і водій (driver). Система забезпечує створення маршрутів, їх моніторинг у режимі реального часу, контроль виконання та обмін даними між учасниками доставки.

Функціональні можливості постачальника наступні:

— створення маршруту;

— призначення водія на маршрут;

— редагування маршруту;

— моніторинг виконання маршруту;

— відмітка про перегляд виконаного маршруту;

— перегляд історії поїздок.

Створення маршруту відбувається введенням опису маршруту, додаванням точок доставки, за потреби додаванням додаткового опису для точки.

Призначення водія відбувається під час створення маршруту.

Редагування маршруту надає можливість додавання, видалення або перестановка точок наявного вже маршруту.

Моніторинг виконання маршруту надає можливість відображення поточного розташування водія на карті та відстеження пройдених і точок, які залишилось пройти.

Перегляд історії поїздок надає повний список завершених маршрутів та можливість повторного створення на основі існуючого.

Функціональні можливості водія наступні:

— доступ до призначеного маршруту;

— навігація та оновлення статусу;

— відображення прогресу;

— завершення маршруту;

— перегляд завершених маршрутів.

Доступ до призначеного маршруту надає можливість перегляду детальної інформації про маршрут перед його початком.

Навігація та оновлення статусу дозволяє маркування точок як "пройдені", автоматичне оновлення координат та визначення наближення до точки з автоматичним оновленням. Також головним алгоритмом є алгоритм знаходження найближчої точки до водія.

Відображення прогресу відбувається візуалізацією на мапі вже відвіданих, активних і майбутніх точок.

Після проходження усіх точок відбувається завершення маршруту, всі потрібні дані відправляються в базу даних.

До загальних можливостей можна віднести:

— аутентифікація користувачів (реєстрація, вхід, перевірка ролі);

— синхронізація даних через Firebase Realtime Database у реальному часі;

— кросплатформена сумісність, завдяки Firebase та Google Maps;

— кешування даних для роботи офлайн і подальшої синхронізації.

Отже, функціональні можливості системи охоплюють повний цикл логістичного процесу — від створення маршрутів до їхнього завершення та звітності. Інтерфейс має бути інтуїтивно зрозумілий як для постачальника, так і для водія, що дозволяє ефективно використовувати застосунок у реальному середовищі доставки.

2.5 Алгоритми роботи ключових модулів

Архітектура проекту передбачає чітке розділення логіки між модулями. Найважливішими є модуль MainActivity, який являється ядром навігації по всьому

застосунку, та NavigationFragment, який відповідальний за маршрутну логіку водія в реальному часі. Нижче подано опис алгоритмів роботи цих модулів.

MainActivity є основною активністю програми та виконує роль контейнера для фрагментів, а також координує навігацію та рольову логіку користувачів. Код модуля наведено в додатку А.

Ключові функції модуля:

— ініціалізація інтерфейсу;

— визначення ролі користувача (supplier/driver) через Firebase Auth;

— завантаження відповідного фрагмента залежно від ролі;

— обробка навігаційного меню та взаємодії користувача з ним;

— перевірка стану авторизації користувача.

Алгоритм дій модуля наведено на рисунку 2.3 [17].

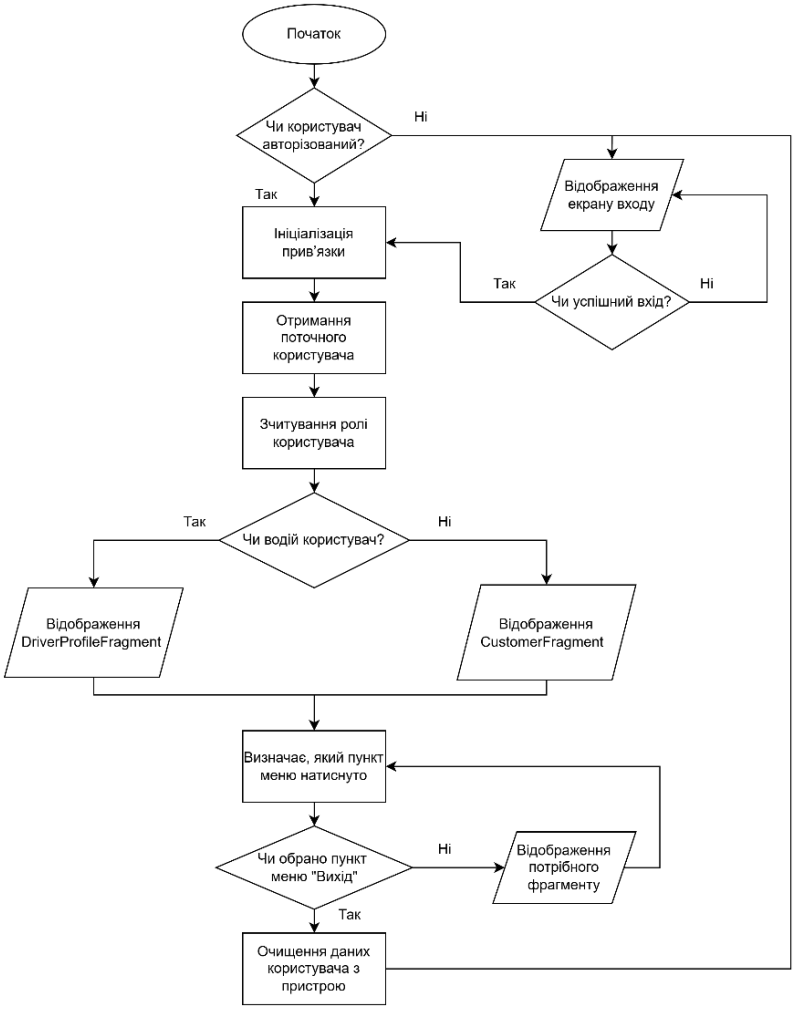

Рисунок 2.3 — Алгоритм дій модуля MainActivity

Розглянемо інтерфейс модуля MainActivity — це файл activity_main.xml. Дизайн інтерфейсу наведено на рисунку 2.4 [18].

activity_main.xml є основним макетом для активності MainActivity та містить наступні елементи:

— <androidx.drawerlayout.widget.DrawerLayout> — контейнер для бокового меню навігації.

— <com.google.android.material.navigation.NavigationView> — відображає меню з пунктами для постачальника або водія.

— <FrameLayout> — динамічний контейнер, у який завантажуються фрагменти в залежності від вибору користувача.

Також використовуються супутні макети.

app_bar_main.xml включає Toolbar.

nav_header_main.xml включає оформлення шапки навігаційного меню, такі якім’я користувача, аватар, email.

content_main.xml включає вміст головного вікна, часто вбудовується у DrawerLayout.

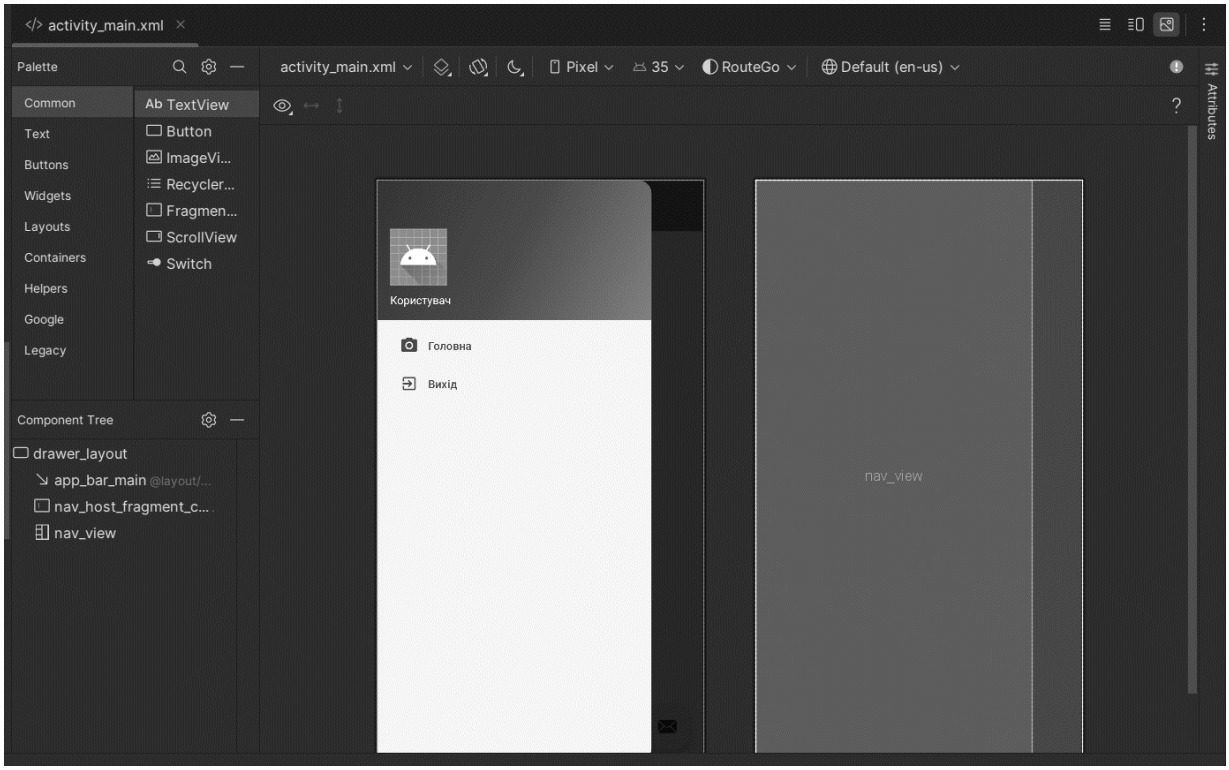

Рисунок 2.4 — Дизайн інтерфейсу activity_main.xml

Модуль NavigationFragment реалізує логіку навігації водія по маршруту, з оновленням координат, візуалізацією карти та контролем відвіданих точок. Код модуля наведено в додатку Б.

Ключові функції модуля:

— отримання та візуалізація списку точок маршруту;

— підключення до карти Google Maps, метод OnMapReadyCallback;

— постійне оновлення місця розташування водія;

— визначення близькості до точок маршруту;

— маркування точок як відвіданих;

— відображення шляху до найближчої точки.

Алгоритм дій модуля наведено на рисунку 2.5.

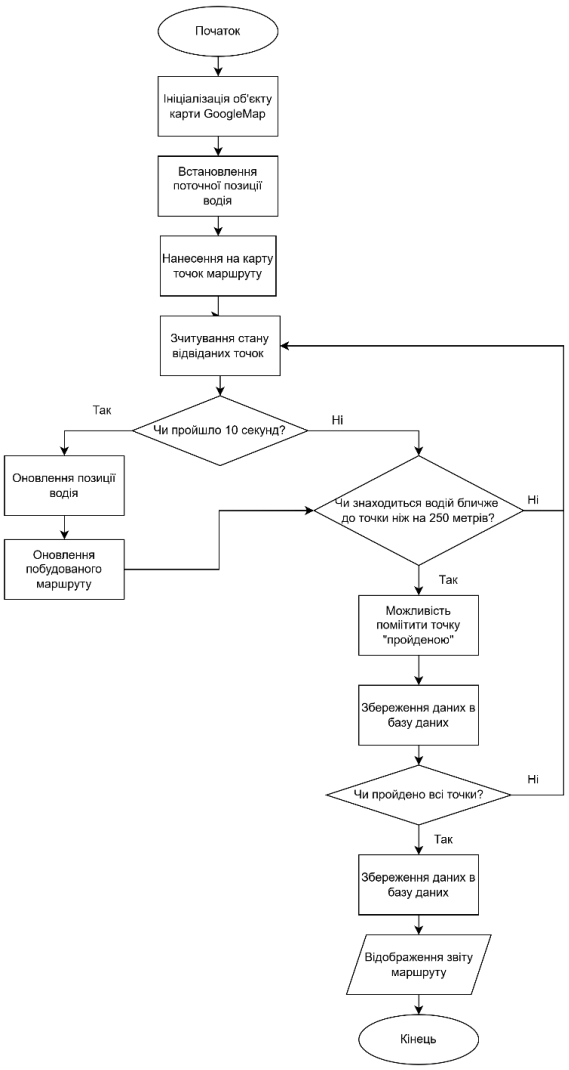

Рисунок 2.5 — Алгоритм дій модуля NavigationFragment

Розглянемо інтерфейс модуля NavigationFragment — це файл fragment_navigation.xml. Дизайн інтерфейсу наведено на рисунку 2.6.

fragment_navigation.xml є основним макетом для активності NavigationFragment та містить наступні елементи:

— <fragment com.google.android.gms.maps.SupportMapFragment > — карта Google Maps.

— <Button> — кнопка, що дозволяє відмітити точку як пройдену.

— <TextView> — текстове поле для відображення інформації про поточну точку маршруту.

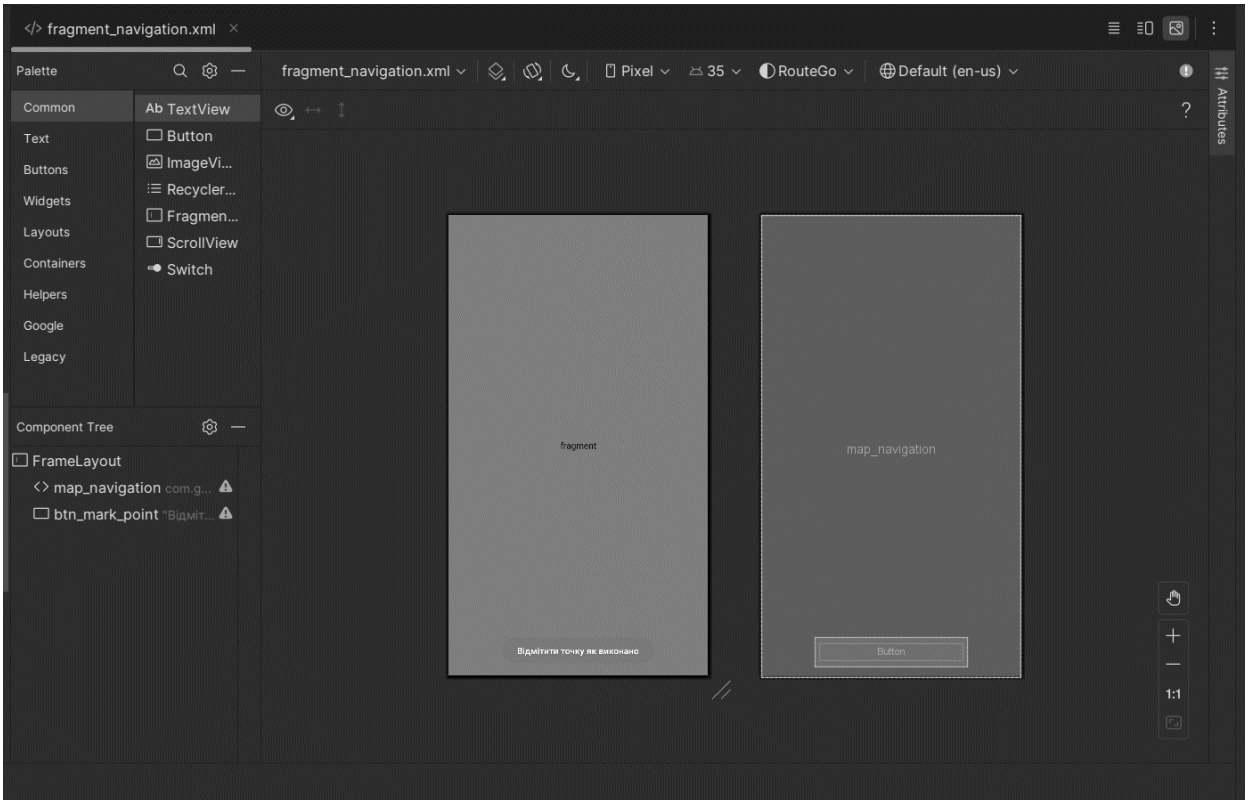

Рисунок 2.6 — Дизайн інтерфейсу fragment_navigation.xml

Модулі MainActivity і NavigationFragment формують ядро логіки взаємодії користувача з системою. MainActivity забезпечує коректну маршрутизацію між фрагментами відповідно до ролі, тоді як NavigationFragment відповідає за процес виконання маршруту, оновлення позиції та взаємодію з картою. Такий розподіл дозволяє досягти високої масштабованості та підтримуваності проекту.

2.6 Апаратне забезпечення мобільного застосунку

Успішне функціонування мобільного застосунку для управління логістикою малого підприємства залежить від апаратних характеристик пристрою, на якому він використовується. Оскільки система має бути реалізована у вигляді Android-застосунку з використанням сервісів реального часу, такими як Firebase, Google Maps API та активною взаємодією з GPS-модулем, важливо визначити технічні характеристики, які забезпечують стабільну роботу всіх компонентів програми.

У таблиці 2.6 подано вимоги мінімальних та рекомендованих параметрів до апаратного забезпечення пристроїв, на яких планується встановлення застосунку [19].

Таблиця 2.6 — Апаратні вимоги до пристроїв

Параметр	**Мінімальні вимоги**	**Рекомендовані вимоги**
Операційна система	Android 8.0 (Oreo)	Android 11 або новіша
Процесор	4-ядерний ARM Cortex A53	8-ядерний ARM Cortex A73 або новіший
Оперативна пам’ять	2 ГБ	4 ГБ або більше
Вбудована пам’ять	16 ГБ	32 ГБ або більше
кран	5.0" з роздільною здатністю 720×1280	6.0"+, роздільна здатність Full HD
GPS-модуль	Обов’язковий, із точністю до 10 м	Високоточний GPS з A-GPS
Підключення до мережі	3G/4G	4G/5G
Доступ до Google Play Services	Обов’язковий	Обов’язковий
Акумулятор	≥ 2500 мА·г	≥ 4000 мА·г

Крім мобільного пристрою користувача, для повноцінного адміністрування, наприклад, управління базою даних Firebase або перегляд логів, може бути

використаний персональний комп’ютер із характеристиками наведеними на таблиці 2.7.

Таблиця 2.7 — Апаратні вимоги до комп’ютера адміністратора

Параметр	**Мінімальні вимоги**	**Рекомендовані вимоги**
Оперативна пам’ять	4 ГБ	8 ГБ
Відеокарта	Інтегрована	Інтегрована / базова дискретна
Дисплей	1366x768	Від Full HD (1920x1080)
Браузер	Google Chrome / Microsoft Edge	Останні версії браузерів з підтримкою DevTools
Швидкість інтернету	20 Мбіт/с	100 Мбіт/с і більше

Отже, визначені апаратні вимоги до пристроїв, на яких буде працювати мобільний застосунок, є доступними для більшості сучасних смартфонів, що робить систему придатною для впровадження у широкому колі малих підприємств. Дотримання рекомендованих параметрів забезпечує стабільну роботу, швидке завантаження інтерфейсу та точну навігацію. Завдяки використанню хмарних технологій не потребується встановлення серверної інфраструктури, що додатково знижує бар’єр для впровадження системи.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Інструментальні засоби та середовища розробки

У процесі реалізації мобільного застосунку для управління логістикою малих підприємств використано сучасні програмні засоби та середовища розробки, що забезпечили ефективне написання коду, тестування, налагодження та запуск проекту на мобільних пристроях і емуляторах. Основними середовищами стали Android Studio та IntelliJ IDEA, а також набір допоміжних інструментів і бібліотек.

Головним середовищем для розробки клієнтської частини системи є Android Studio. Android Studio являє собою офіційну IDE для створення Android-застосунків. У проекті використовувалася версія Android Studio Electric Eel, яка підтримує мову програмування Java, інтеграцію з Google Maps SDK, Firebase та візуальний редактор інтерфейсів.

Android Studio надала повноцінний набір функцій: автодоповнення коду, візуалізацію макетів, симулятори пристроїв, а саме AVD Manager, засоби профілювання продуктивності, а також інтеграцію з системою керування версіями [20].

Окрім Android Studio, частина роботи над проектом, редагуванням допоміжних Java-класів та тестуванням логіки виконувалась у IntelliJ IDEA. Це середовище, яке забезпечило швидку навігацію по класах, рефакторинг коду, зручну підтримку бібліотек та інтеграцію з Maven і Gradle. IntelliJ IDEA також використовувалась для перевірки логіки обробки запитів до Firebase і розділення ролей користувачів. Графічний інтерфейс IntelliJ IDEA навдено на рисунку 3.1 [21].

Для реалізації карти маршруту в застосунку було використано Google Maps SDK for Android, що забезпечив інтерактивне відображення маршрутів, маркерів точок доставки та позиції водія. API дозволив програмно керувати картою, додавати маркери, будувати полігональні лінії та реагувати на географічні координати користувача.

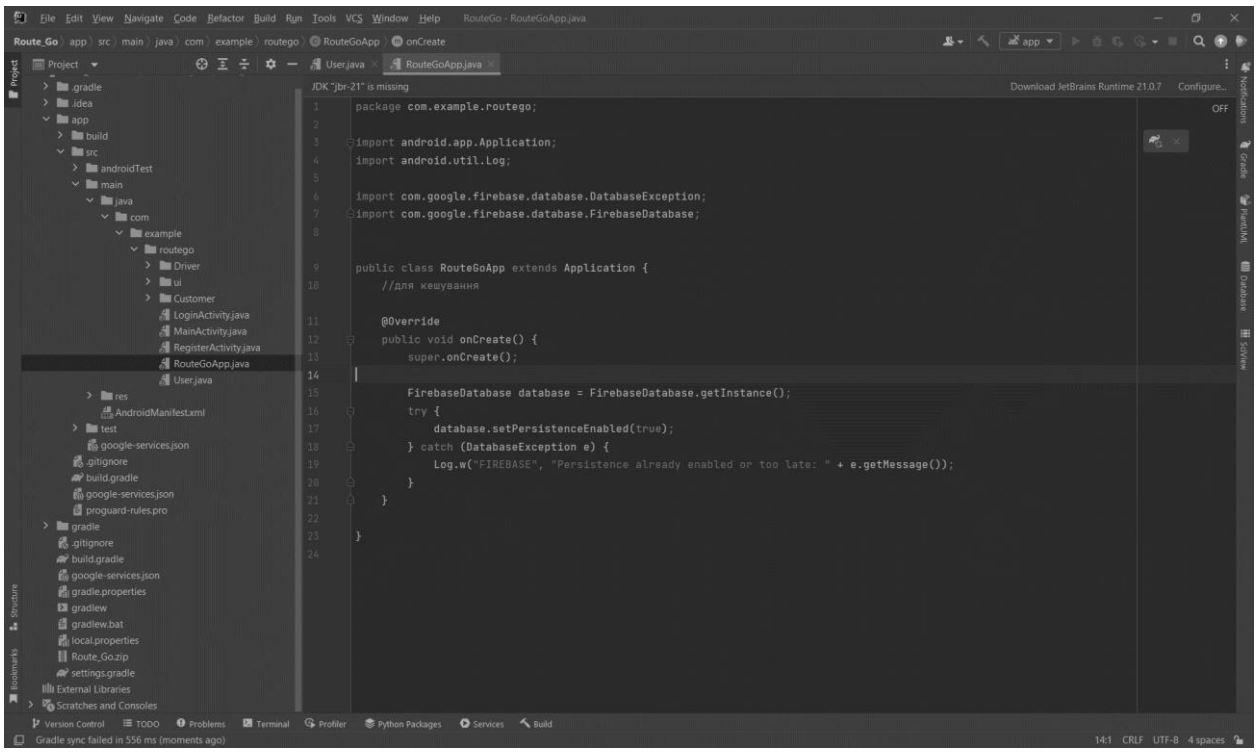

Рисунок 3.1 — Графічний інтерфейс IntelliJ IDEA

Для зберігання даних, аутентифікації та обміну повідомленнями між користувачами було використано хмарну платформу Firebase. Зокрема, застосовано наступні сервіси:

— Firebase Authentication — для реєстрації та входу користувачів;

— Firebase Realtime Database — для зберігання інформації про маршрути, користувачів і координати;

— Firebase Console — як засіб адміністрування бази даних та моніторингу активності.

Для графічного дизайну окремих компонентів інтерфейсу використовувалися стандартні інструменти Android SDK, а також XML-розмітка макетів у редакторі Layout Editor. При налагодженні карти маршруту та роботи з геолокацією застосовувався Google API Emulator, що дозволив змоделювати рух водія без фізичного переміщення пристрою.

Для тестування застосунку на фізичному пристрої використовувався смартфон Redmi Note 8 Pro, а також вбудований Android Emulator з конфігурацією

Android 15.0, що надасть змогу перевірити стабільність роботи в різних середовищах та при різних умовах.

Отже, обрана сукупність інструментів дозволила повністю реалізувати функціонал, передбачений вимогами до мобільного застосунку, та забезпечити його зручне розгортання, масштабування і підтримку.

3.2 Розробка серверної частини та API

Серверна частина мобільного застосунку реалізована на базі хмарної платформи Firebase, яка виконує роль backend-системи та бази даних. Графічний інтерфейс сторінки Firebase наведено на рисунку 3.2 [22].

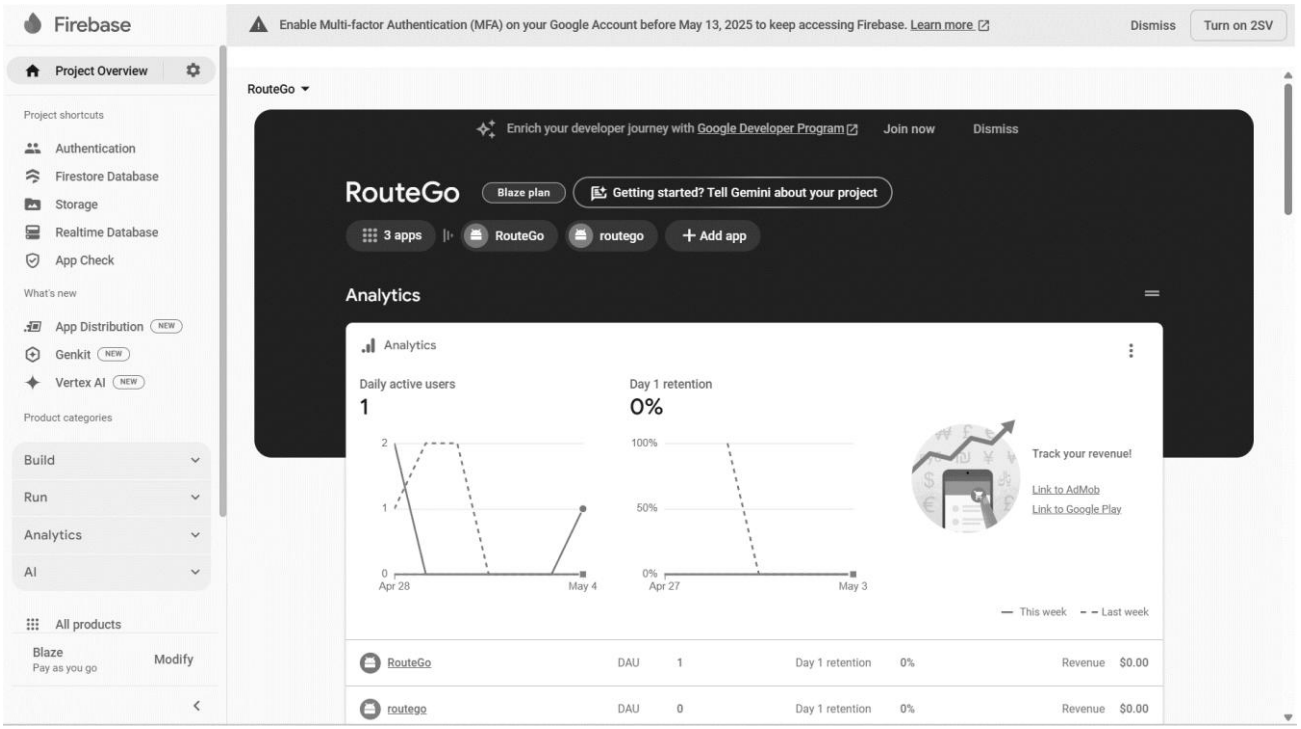

Рисунок 3.2 — Графічний інтерфейс сторінки Firebase

Це рішення було обране тому що дозволяє швидко впровадити вже в наявний проект, за потреби відбувається автоматичне масштабування, не потрібно розгортати власний сервер та має безкоштовний тариф для початкового використання. Окрім цього варто звернути увагу на вбудовані механізми безпеки та аутентифікації.

Компоненти серверної частини:

— Firebase Authentication;

— Firebase Realtime Database;

— Firebase SDK for Android.

Firebase Authentication відповідає за реєстрацію, вхід та перевірка прав доступу користувачів.

Firebase Realtime Database являє собі базу даних, яка працює в реальному часі, всі зміни відразу можна побачити, без додаткових оновлень. Зберігає інформацію про користувачів, маршрути, координати водіїв.

Уся взаємодія клієнта з сервером відбувається через Firebase SDK, який надає функціональність без потреби створювати REST-ендпоінти вручну. Проте можна інтерпретувати такі дії як логічні API-запити.

Головною перевагою використання Firebase є повна інтеграція з Android. Вбудована безпека доступу, захищає додаток від несанкціонованого доступу.

Проте існує ряд обмежень, таких як, складність збереження складних зв’язків, немає SQL JOIN, платна масштабована версія при великій кількості користувачів, обмеження при офлайн-режимі, компенсується кешуванням через setPersistenceEnabled(true)).

Отже, Firebase як серверна частина надає ряд переваг та позбавляє необхідності створювати окремий backend, такі компоненти як сервер, база даних, API. Простота розгортання і підтримки дозволяє розмістити систему під мале підприємство з обмеженими ресурсами. Швидкий старт без розгортання додаткової інфраструктури, дозволяє пришвидшити процес початку роботи.

3.3 Реалізація клієнтської частини мобільного застосунку

Клієнтська частина мобільного застосунку розроблена з використанням Android SDK у середовищі Android Studio мовою програмування Java.

Архітектура побудована з урахуванням розділення логіки за ролями користувачів: водій та постачальник, а також підтримки карти, геолокації, реєстрації маршрутів і їх оновлення в реальному часі.

Використано Firebase Auth для аутентифікація користувачів, обрано вхід через email/пароль. Ця технологія дозволяє звертатися до Firebase та отримувати дані щодо входу і чи наявний потрібний користувач.

Технологія Firebase Realtime Database використовується для зберігання маршрутів, користувачів, координат та отримання цих даних за потребою.

Google Maps SDK for Android відображає мапу Google Maps та робить її інтерактивною. Щоб вперше налаштувати Google Maps SDK for Android у своєму Android-проекті, потрібно в Google Cloud Console створити новий проект та додати використання Google Maps Platform за допомогою. Після додавання ми побачимо графічний інтерфейс Google Cloud Console, яке наведено рисунку 3.3 [23].

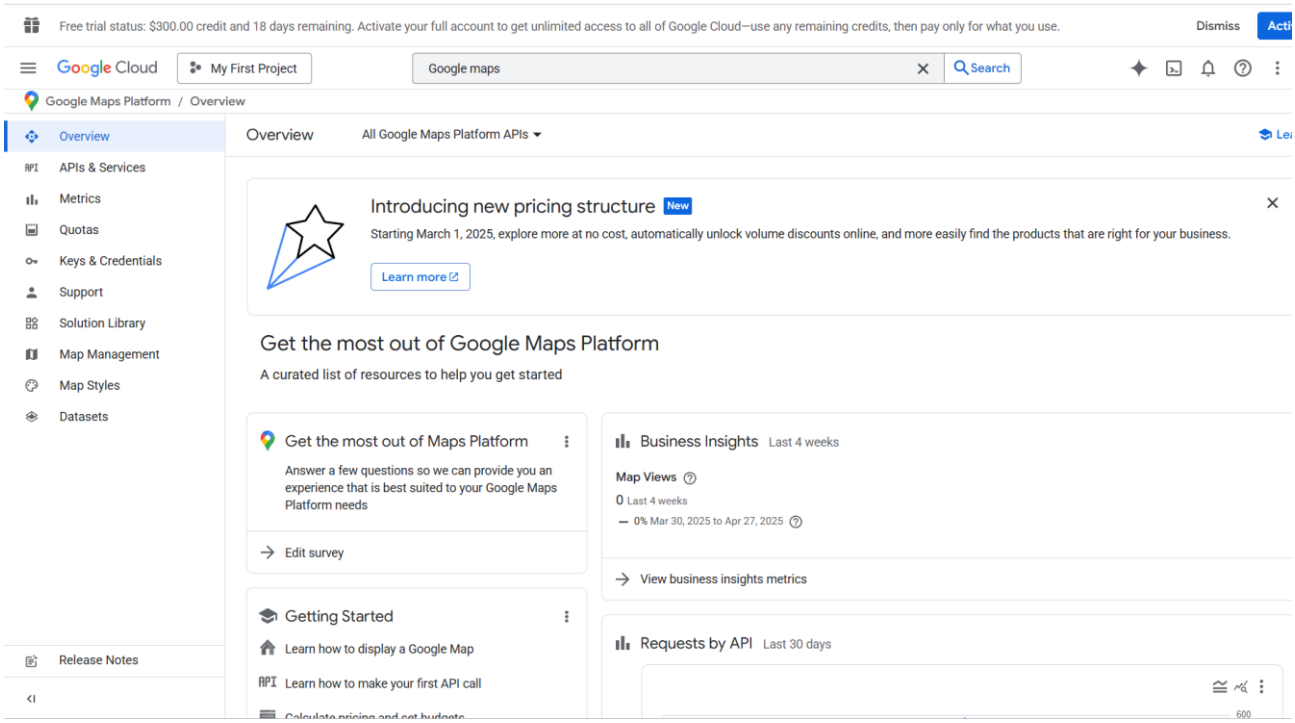

Рисунок 3.3 — Графічний інтерфейс Google Cloud Console

Далі треба створити Credentials, це ключі з якими буде працювати додаток. Як ключі створено ми побачимо список всіх API keys, яке наведено рисунку 3.4.

Далі, необхідний API key треба завантажити та додати в папку з проектом. Після цього треба налаштувати всі залежності в build.gradle та налаштувати дозволи на локації та передачу даних. Для цього перейдемо в файл

AndroidManifest.xml, який відповідає за дозволи проекту та додаємо наступні стрічки навдені у лістингу 3.1

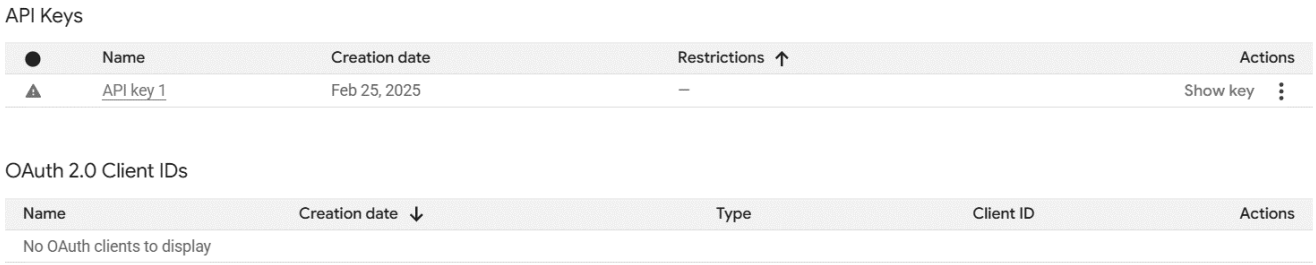

Рисунок 3.4 — Список всіх API keys

Лістинг 3.1 — Зміни в AndroidManifest.xml

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Тепер можемо використовувати Google Maps, для цього на інтерфейсі створимо фрагмент з картою, який наведено на лістингу 3.2.

Лістинг 3.2 — Фрагмент Google Maps

```
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Основні компоненти клієнтської частини є: LoginActivity.java, RegisterActivity.java, MainActivity.java, NavigationFragment.java, SupplierMapFragment.java / SupplierProfileFragment.java.

LoginActivity.java відповідає вхід користувача з валідацією полів email та пароль. Авторизація проходить через

FirebaseAuth.signInWithEmailAndPassword(...). В залежності від результату визначається роль користувача і перенаправляється на відповідний інтерфейс. Графічний інтерфейс LoginActivity наведено на рисунку 3.5.

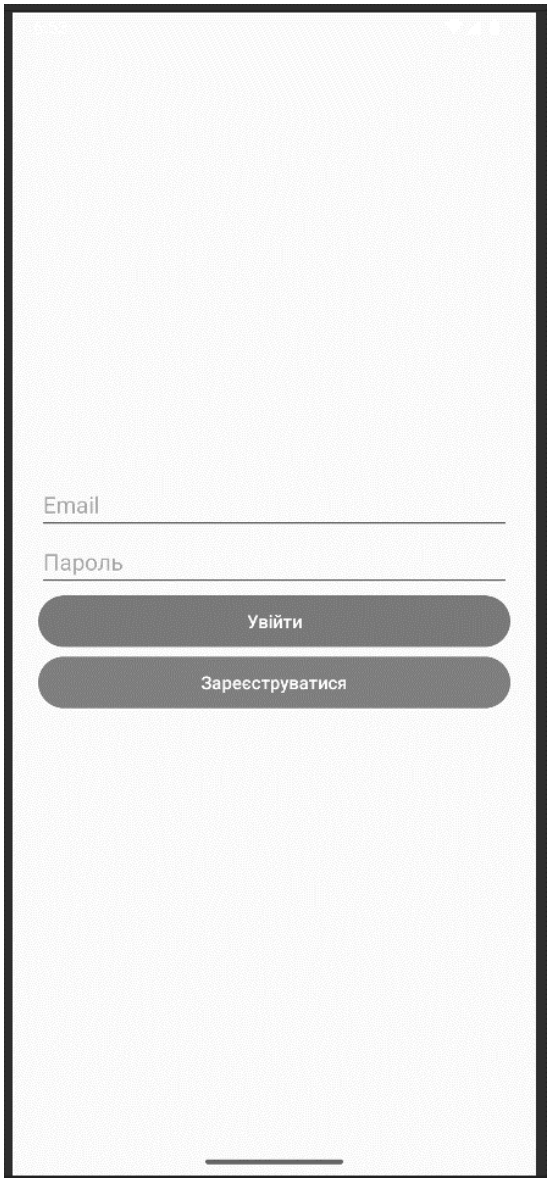

Рисунок 3.5 — Графічний інтерфейс LoginActivity

RegisterActivity.java відповідає за реєстрацію нового користувача та додавання запису до Users з полями role, isApproved, email. Графічний інтерфейс RegisterActivity наведено на рисунку 3.6.

MainActivity.java це контейнер для основного інтерфейсу. Відображає фрагменти залежно від ролі користувача, для водія — DriverRoutesFragment, для постачальника — SupplierProfileFragment.

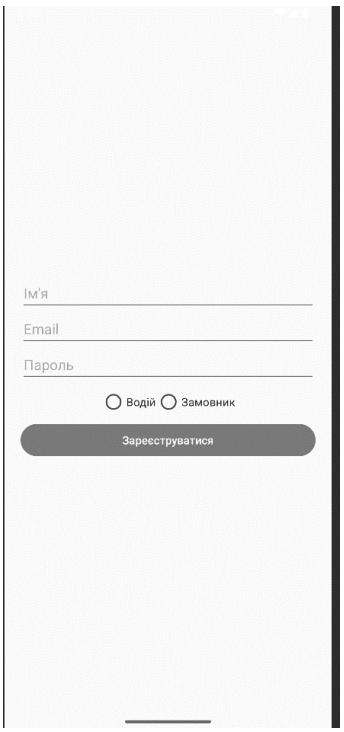

Рисунок 3.6 — Графічний інтерфейс RegisterActivity

Фрагмент SupplierProfileFragment постачальнику відображає активні поїдки та водіїв на мапі, оновлення координат відбувається у реальному часі. Графічний інтерфейс SupplierProfileFragment наведено на рисунку 3.7.

Рисунок 3.7 — Графічний інтерфейс SupplierProfileFragment

Фрагмент DriverRoutesFragment відображає поточні поїдки водія. Графічний інтерфейс DriverRoutesFragment наведено на рисунку 3.8.

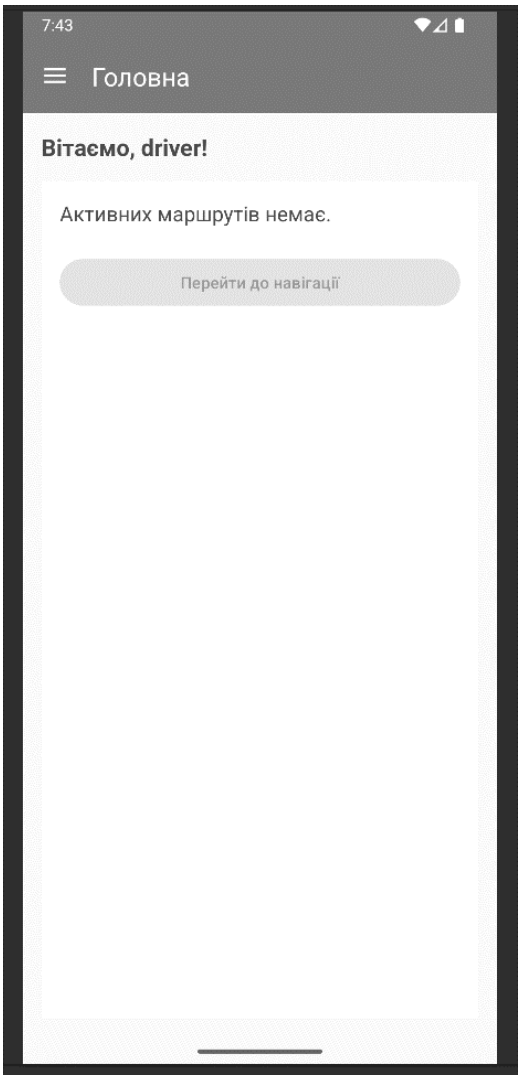

Рисунок 3.8 — Графічний інтерфейс DriverRoutesFragment

Розглянемо алгоритм початку маршруту. У фрагменті DriverRoutesFragment, який завантажується після авторизації водія, система виконує пошук активного маршруту в базі даних Firebase серед тих, де ідентифікатор водія збігається з поточним користувачем, а статус маршруту дорівнює "active". Якщо такий маршрут знайдено, застосунок завантажує список усіх точок маршруту та визначає першу точку, яка ще не була відмічена як пройдена. Графічний інтерфейс NavigationFragment наведено на рисунку 3.9.

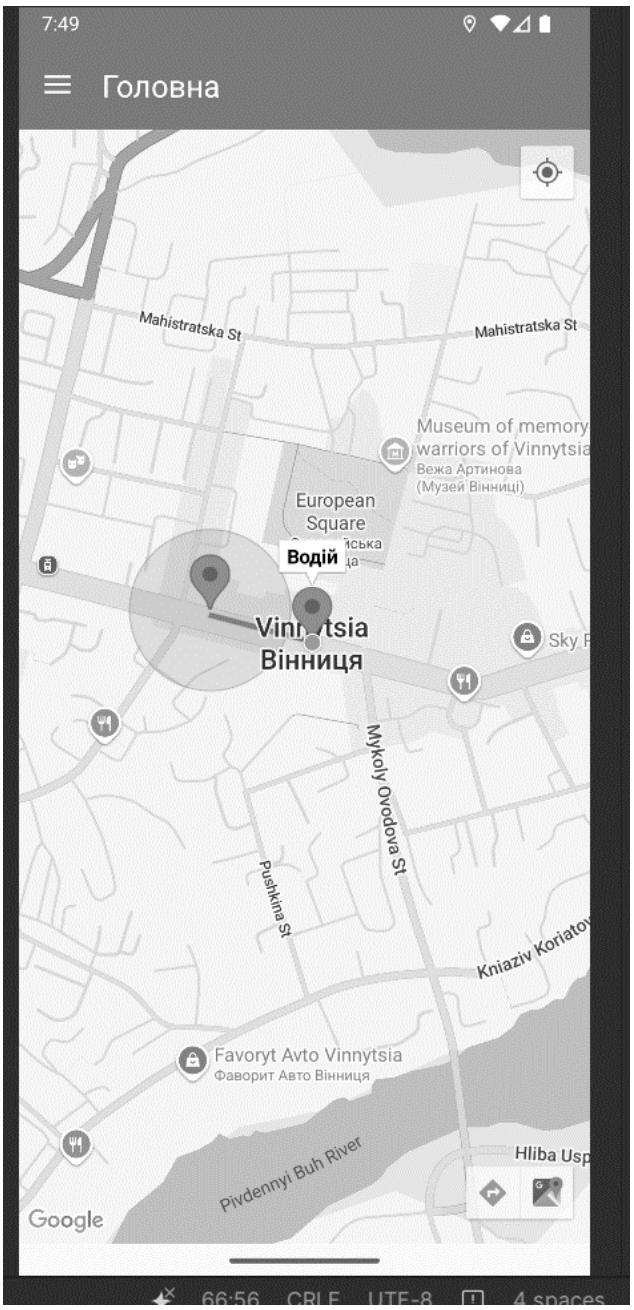

Рисунок 3.9 — Графічний інтерфейс NavigationFragment

Після цього карта ініціалізується за допомогою та усі точки маршруту відображаються на карті маркерами, при цьому пройдені точки мають один колір, а не пройдені – інший. Також, запускається механізм геолокації, щоб отримувати координати пристрою водія з певною. Кожне нове значення координат відправляється до бази даних Firebase у розділ DriversLocation, звідки постачальник може отримати місцезнаходження водія.

При кожному оновленні координат виконується перевірка — чи наблизився водій до цільової точки. Після підтвердження система переходить до наступної точки маршруту, яка ще не пройдена. Якщо така точка є, тоді застосунок будує маршрут. Якщо ж усі точки вже пройдено, тоді маршрут вважається завершеним, та користувач бачить повідомлення про завершення маршруту, і робота фрагмента закінчується.

Якщо поточний маршрут не знайдено, необхідно вибрати з наявних. Для цього необхідно з меню вибрати «Доступні замовлення». Графічний інтерфейс «Доступних маршрутів» наведено на рисунку 3.1.

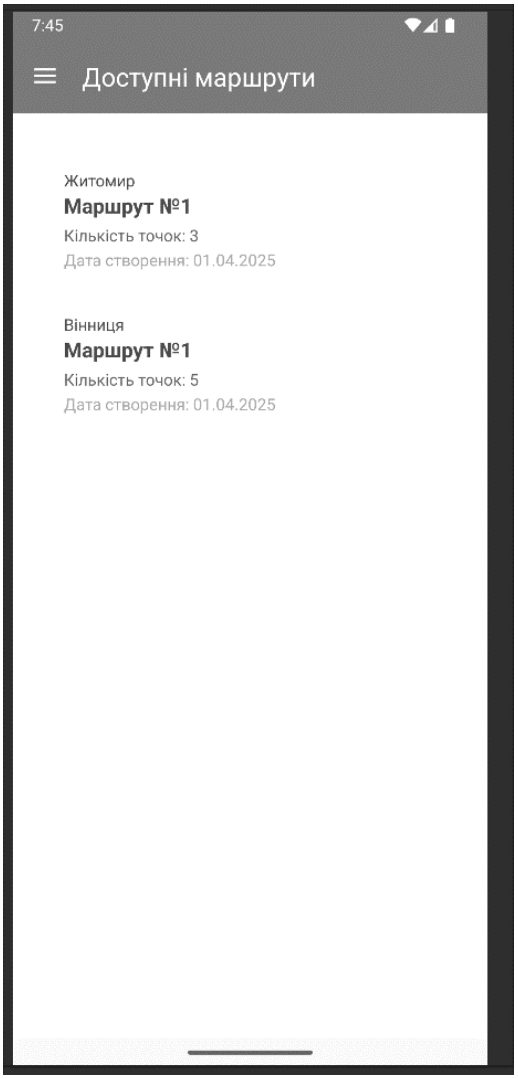

Рисунок 3.10 — Графічний інтерфейс «Доступних маршрутів»

Серед наявних вибрати та переглянути потрібний. Якщо всі умови влаштовують, тоді необхідно підвередити початок, і відбудеться перехід до

поточного маршруту. Графічний інтерфейс «Перегляду маршруту» наведено на рисунку 3.11.

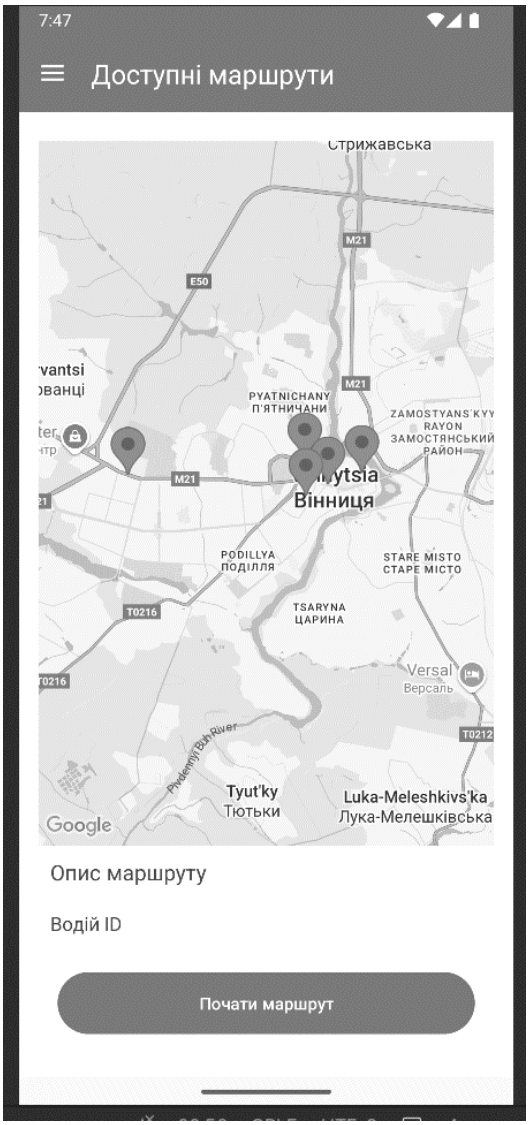

Рисунок 3.11 — Графічний інтерфейс «Перегляду маршруту»

Інтерфейс розроблений у мінімалістичному стилі, з урахуванням адаптивності для різних екранів, українською мовою.

Також задля оптимізації геолокація оновлюється тільки при русі, щоб економити трафік. Всі запити до Firebase хешуються (setPersistenceEnabled(true)), щоб при втраті підключення зберегти дані. Взаємодія з картою працює асинхронно, з обробкою помилок (OnMapReadyCallback).

Отже, клієнтська частина мобільного застосунку реалізована з урахуванням сучасних вимог до зручності, надійності та адаптивності. Завдяки використанню

Android SDK, інтеграції з Firebase та Google Maps, вдалося забезпечити повноцінну взаємодію користувача з системою. Водії отримують доступ до маршрутів, оновлюють своє місцезнаходження та фіксують проходження точок у реальному часі, тоді як постачальники мають змогу створювати маршрути, контролювати їх виконання та переглядати історію поїздок.

3.4 Опис взаємодії між компонентами системи

Мобільний застосунок побудований на основі компонентно-орієнтованої архітектури Android, що включає активності, фрагменти, сервіси та Firebase-сервіси. Взаємодія між компонентами клієнтської частини та хмарними сервісами реалізується асинхронно з використанням Firebase SDK, що забезпечує оновлення даних у реальному часі. Схема взаємодія користувачів з компонентами наведено на рисунку 3.12.

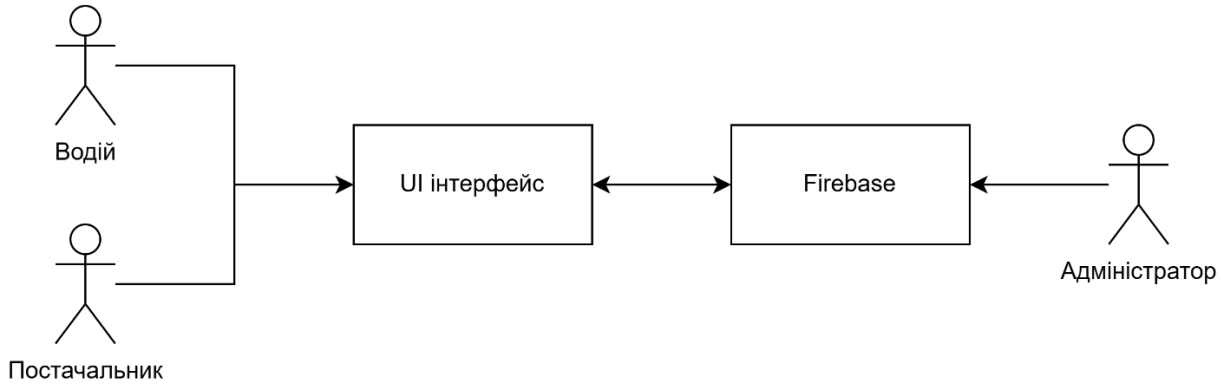

Рисунок 3.12 — Схема взаємодія користувачів з компонентами

Користувачі взаємодіють із застосунком через активності та фрагменти, такі як LoginActivity, RegisterActivity, MainActivity, NavigationFragment, SupplierMapFragment, SupplierProfileFragment. Усі ці компоненти реалізують логіку відображення інтерфейсу, обробки натискань, перевірки прав доступу та викликів до бази даних.

UI-компоненти викликають відповідні методи Firebase, наприклад аутентифікація через FirebaseAuth та зчитування з записом користувачів, маршрутів, координат — через FirebaseDatabase.

Водійська частина застосунку постійно передає координати водія в базу даних, використовуючи сервіси геолокації. Координати надсилаються до колекції DriversLocation, звідки вони доступні для постачальника у режимі реального часу.

Взаємодія комопнентів побудована за принципами реактивної архітектури, де зміни у базі даних одразу спричиняють оновлення стану в UI.

Реактивна архітектура — це прийнятий підхід до побудови програмного забезпечення, у якому система реагує на події та зміни стану у реальному часі. В застосунку реактивна архітектура реалізується через Firebase Realtime Database та автоматичне оновлення UI при зміні даних у базі.

Отже, створена модель взаємодії дозволяє досягти максимальної прозорості у відображенні логістичних процесів, зменшити кількість запитів до сервера та забезпечити високий рівень зручності як для водіїв, так і для постачальників.

3.5 Впровадження механізмів аутентифікації та безпеки даних

Безпека даних та контроль доступу є ключовими аспектами при розробці мобільного застосунку, особливо в логістичних системах, де обробляється конфіденційна інформація, наприклад маршрути, геолокація водіїв і персональні дані користувачів.

Для забезпечення надійного входу та реєстрації в систему використано Firebase Authentication, який підтримує просту інтеграцію та безпечне зберігання облікових даних.

Після створення облікового запису, в базу даних Realtime Database автоматично додається запис про нового користувача з інформацією про його роль та статус підтвердження. Доступ до основних функцій застосунку можливий лише після проходження перевірки. Перевірку здійснює адміністратор Firebase, в базі даних перевіряє дані на достовірність, якщо все добре, вводить параметр isApproved = true.

Після авторизації застосунок визначає роль користувача та на її основі формує інтерфейс: водій переходить до поточного маршруту, а постачальник — до інтерфейсу керування маршрутами та відстеженням водіїв. Усі дії, пов’язані з

читанням або записом даних у базу, супроводжуються перевіркою прав доступу, які визначаються у спеціальних правилах безпеки Firebase.

Правила безпеки Firebase можливо керувати з консолі у форматі JSON. Консоль керування правилами безпеки Firebase наведено на рисунку 3.13.

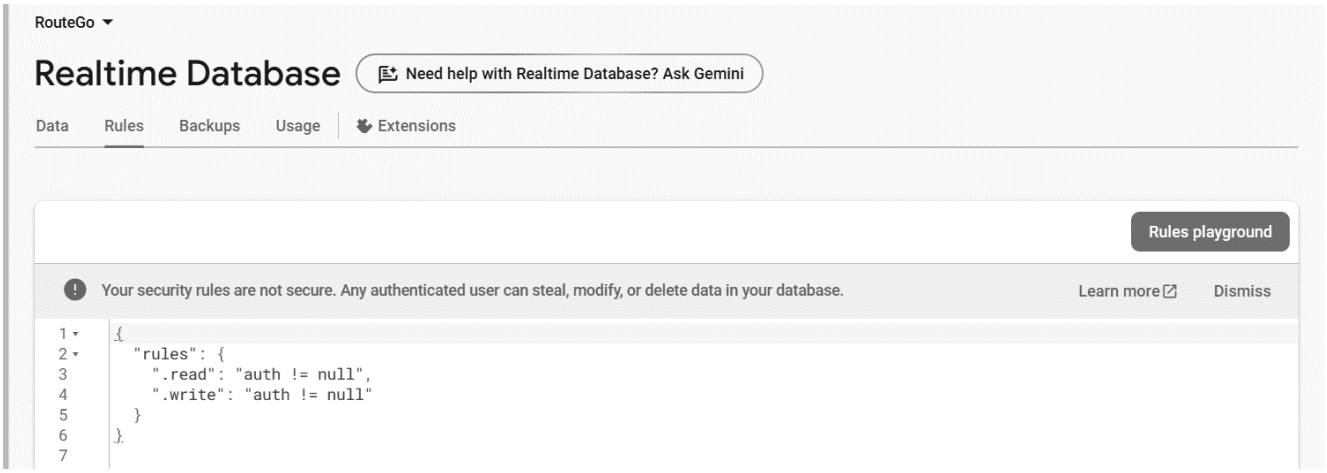

Рисунок 3.13 — Консоль керування правилами безпеки Firebase

Поточні правила гарантують, що до бази можуть звертатись лише авторизовані користувачі, і виключають можливість стороннього доступу до даних. Передача інформації між клієнтом і сервером здійснюється через захищений протокол HTTPS, що забезпечує шифрування даних і захист від несанкціонованого перехоплення.

Особливу увагу приділено контролю доступу до даних у межах самої системи. Кожна дія супроводжується ідентифікацією користувача за унікальним UID, що унеможливлює доступ водія до чужого маршруту. Навіть у випадку отримання доступу до коду застосунку, скористатися ним без авторизації та проходження перевірки прав буде неможливо. Для додаткового захисту використовуються механізми обмеження Google Maps API за підписом додатку, що унеможливлює використання API-ключа сторонніми особами, тому що ключ прив’язаний саме до застосунку.

Отже, система автентифікації та захисту даних у застосунку реалізована комплексно. Вона поєднує безпечну авторизацію, гнучку перевірку прав доступу та зашифровану передачу даних.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

4.1 Тестування функціоналу мобільного застосунку

Тестування функціоналу мобільного застосунку проводилось з метою перевірки коректності реалізації основних сценаріїв використання системи та виявлення потенційних помилок у логіці взаємодії з Firebase Realtime Database nf Google Maps API.

Для тестування використано реальний пристрій, Xiaomi Redmi Note 8 Pro та емулятор Android Studio з Android API 35, що дозволило здійснити повне покриття функціональних гілок програми та оцінити поведінку в різних умовах.

Функціональне тестування включало перевірку всіх етапів взаємодії користувача з системою, починаючи від реєстрації нового облікового запису та завершуючи успішним проходженням маршруту водієм. Кожен із наступних модулів був протестований окремо та у зв’язку з іншими компонентами системи.

В першу чергу протестовано реєстрацію та вхід до системи. У процесі реєстрації застосунок перевіряє валідність електронної пошти та паролю, створює обліковий запис у Firebase Authentication, а також записує інформацію про нового користувача до колекції Users у базі даних, де зберігається його роль та статус підтвердження.

Після входу система коректно визначає роль користувача і спрямовує його до відповідного інтерфейсу. Протестовано поведінку системи при введенні неправильних даних, при незаповнених полях, а також при спробі входу користувача з неактивним обліковим записом – результат задовольняє очікування.

Другим етапом стало тестування створення маршруту постачальником. У фрагменті профілю постачальника реалізовано функціонал створення маршруту з кількома точками доставки. Було перевірено коректність побудови маршруту на мапі, правильне збереження кожної точки у Firebase та присвоєння маршруту статусу active.

Також було протестовано механізм редагування маршруту, повторне використання збережених маршрутів та передачу маршруту конкретному водієві.

Особливу увагу приділено тестуванню функціоналу NavigationFragment, де відображається активний маршрут для водія. Після входу водія в систему, застосунок правильно зчитує маршрут із бази даних, фільтрує точки, які ще не пройдені, та будує на їх основі маршрут на мапі з використанням Google Maps SDK.

Окремо перевірялася точність фіксації координат водія та їх відправка до Firebase. В процесі проходження маршруту тестувалася система автоматичного визначення досягнення точки, при наближенні менше ніж 250 метрів, після чого користувач помічав точку поміченою та відбувався перехід до наступної.

Також перевірено сценарій завершення маршруту, коли всі точки позначені як пройдені, маршрут змінює свій статус на completed. Було протестовано поведінку системи при навмисному виході з програми або втраті інтернет-з'єднання, проте усі ключові дані залишаються збереженими завдяки використанню офлайн-кешу Firebase.

Крім позитивних сценаріїв, було проведено тестування негативних випадків, таких як спроба водія отримати маршрут без призначеного маршруту, створення маршруту без точок, входження до системи без дозволів на геолокацію та відсутність інтернет-з'єднання на момент авторизації.

Усі критичні помилки були успішно виявлені й усунуті, що дозволило забезпечити стабільну роботу застосунку в типових умовах експлуатації.

Отже, функціональне тестування підтвердило, що застосунок правильно реалізує основні бізнес-процеси логістики малого підприємства. Застосунок є готовим до використання в реальному середовищі, забезпечуючи передбачувану поведінку в більшості сценаріїв, включаючи обробку виняткових ситуацій.

4.2 Перевірка продуктивності та оптимізація мобільного застосунку

Продуктивність мобільного застосунку відіграє ключову роль у забезпеченні комфортного користувацького досвіду, особливо в умовах, коли застосунок активно взаємодіє з картою, оновлює координати в реальному часі та виконує часті запити до бази даних. У межах оцінки продуктивності було протестовано

швидкодію, використання ресурсів пристрою, час реакції інтерфейсу, а також стабільність роботи застосунку за умов тривалого використання.

Перш за все було виміряно час запуску основних фрагментів після входу в систему.

На реальному пристрої, Xiaomi Redmi Note 8 Pro перехід від LoginActivity до MainActivity із підвантаженням відповідного фрагмента відбувався в межах 1.2–1.5 секунд, включаючи зчитування ролі користувача з Firebase. Оскільки система не використовує важкої серверної логіки, усі дії відбуваються асинхронно з мінімальними затримками, що забезпечує майже миттєву реакцію інтерфейсу на дії користувача.

У процесі навігації водія було перевірено швидкість оновлення координат. Визначення поточної позиції здійснюється через FusedLocationProviderClient з інтервалом в 10 секунд. Всі координати надсилаються в Realtime Database, де зчитуються постачальником. Затримка між оновленням координат і їхнім відображенням на карті в інтерфейсі постачальника становить в середньому менше 1 секунди, що свідчить про високу ефективність обміну даними через Firebase.

Особлива увага була приділена оптимізації взаємодії з Firebase. У застосунку використовуються лише необхідні запити до бази даних, які чітко обмежені за шляхами, таким як Users, Routes, DriversLocation. Не здійснюється надмірне спостереження за великими гілками бази, що дозволяє зменшити навантаження на мережу та оперативну пам’ять.

Було також враховано правила використання setPersistenceEnabled(true), яке задається у класі RouteGoApp до першого звернення до бази. Це дозволило запобігти помилкам ініціалізації, з якими система стикалася на ранніх етапах та дозволяє зберігати дані в кеші програми.

У Google Maps API було оптимізовано відображення точок маршруту. Замість постійного перерахунку маршруту після кожного оновлення геолокації, маршрут будується лише один раз при ініціалізації фрагмента, а динамічно змінюється лише маркування активної точки. Це дозволяє уникнути надмірного

споживання ресурсів графічного процесора, що особливо важливо для бюджетних Android-пристроїв.

Загалом продуктивність застосунку визнана задовільною. Застосунок стабільно працює протягом тривалого часу, без витоків пам’яті чи збоїв. Тестування на емуляторі виявило потенційні проблеми з продуктивністю при високому навантаженні на графічні ресурси, проте вони не відтворюються на реальному пристрої. Це свідчить про те, що застосунок орієнтований на реальні умови використання, де демонструє високу чутливість до дій користувача, швидке завантаження і ефективну роботу з ресурсами системи.

Отже, за результатами оцінки продуктивності можна стверджувати, що застосунок відповідає вимогам малих підприємств до швидкодії, стабільності та ефективного використання мережевих ресурсів.

4.3 Інструкція користувача

Мобільний застосунок розроблено з урахуванням потреб двох основних ролей користувачів – постачальника, відповідального за створення та керування маршрутами, та водія, який виконує доставку згідно із заданим маршрутом. Інтерфейс застосунку побудований таким чином, щоб забезпечити максимально просту та інтуїтивну взаємодію, навіть для користувачів без спеціальної технічної підготовки.

Після встановлення застосунку користувач має можливість пройти реєстрацію. Для цього необхідно вказати електронну пошту, пароль, ім’я, номер телефону та вибрати одну з ролей: водій або постачальник. Після завершення реєстрації обліковий запис потрапляє в базу даних. Система вимагає підтвердження акаунту з боку адміністратора. До моменту схвалення, при спробі входу користувач отримає повідомлення про те, що його акаунт ще не підтверджено.

Після успішного входу до системи інтерфейс налаштовується залежно від ролі. Постачальник потрапляє до особистого кабінету, де бачить список активних маршрутів, історію поїздок та кабінет постачальника. В меню є кнопка «Створити маршрут», яка відкриває форму додавання нового маршруту. Постачальник може

додати декілька точок доставки, обрати з випадаючого списку водія, вказати назву маршруту та підтвердити створення. Після цього маршрут зберігається в базі та стає доступним призначеному водієві.

У разі необхідності повторного використання вже завершеного маршруту, постачальник може обрати його з історії та натиснути кнопку «Повторити». Це відкриє форму редагування, де можна змінити точки або дані про водія, після чого створиться новий активний маршрут на основі попереднього.

Водій, після входу в систему, автоматично перенаправляється до фрагмента активного маршруту. Фрагмент перевіряє чи є активний маршрут, закріплений за поточним користувачем. Якщо маршрут знайдено, тоді надається доступ до відстеження, де карта відображає всі точки маршруту, а також поточну цільову точку, яку потрібно відвідати. Система автоматично визначає відстань до точки, і коли водій наближається до неї на відстань менше або рівну 250 метрів, точка може бути позначена як пройдена. Далі система автоматично переходить до наступної точки маршруту.

На екрані водія завжди відображається актуальна інформація: назва поточної точки та карта з позначками. Крім того, водій не має потреби вручну оновлювати свій маршрут, тому що дані синхронізуються з Firebase у реальному часі. У разі втрати підключення координати тимчасово зберігаються локально і надсилаються після відновлення з’єднання.

Після проходження всіх точок маршруту остання точка автоматично завершує маршрут. Водій бачить повідомлення з підтвердженням завершення роботи.

Отже, застосунок реалізує повний цикл логістичної доставки, від створення маршруту до його виконання. Завдяки простому інтерфейсу, логічній структурі дій та наочному виведенню інформації, застосунок забезпечує зрозумілу взаємодію для користувачів будь-якого рівня підготовки.

4.4 Можливості для подальшого розвитку та масштабування

Мобільний застосунок, на поточному етапі надає повноцінний

інструментарій для управління логістичними процесами на рівні малих підприємств. Проте з оглядом на динамічний розвиток цифрових технологій та потенціал зростання обсягу перевезень, застосунок має широкі можливості для подальшого розвитку та масштабування.

По перше, подальший розвиток може передбачати розширення ролей користувачів. У поточній версії реалізовано всього дві основні ролі: постачальник і водій. Для більших організацій доречним буде впровадження ролі диспетчера, який зможе керувати кількома маршрутами, оперативно перенаправляти водіїв та комунікувати з клієнтами. Також можлива розробка адміністративного вебінтерфейсу, який дозволить зручно керувати даними з персонального комп’ютера або планшета.

Застосунок може бути масштабований до підтримки кількох паралельних маршрутів на одного водія, а також до реалізації більш складних логістичних задач. Такий функціонал може бути реалізований через інтеграцію із зовнішніми алгоритмами оптимізації маршрутів.

Важливим напрямом розвитку є розширення комунікації між учасниками процесу. У майбутнього можливе додавання вбудованого чату між водієм і постачальником, сповіщень про форс-мажорні обставини, або навіть інтеграції з популярними месенджерами для миттєвої передачі критичної інформації.

З огляду на зростання обсягу даних із часом, доцільним є створення аналітичного модуля, що дозволятиме будувати звіти за період, обчислювати середній час доставки, кількість виконаних маршрутів, затримки та інші ключові показники. Такі функції не лише підвищать прозорість, але нададуть керівництву можливість приймати обґрунтовані управлінські рішення.

Технічно архітектура застосунку вже підтримує масштабування. Firebase автоматично обробляє збільшення навантаження, а структура бази даних дозволяє додавати нові гілки. Клієнтська частина легко адаптується до змін у структурі даних, що дозволяє поступово нарощувати функціонал без повного переписування коду.

Отже, розроблений застосунок має не лише завершений функціональний мінімум, а й стратегічний потенціал для подальшого розвитку. Його гнучка архітектура, інтеграція з хмарними технологіями та орієнтація на практичні потреби логістики відкривають широкі перспективи для адаптації до різних сценаріїв, бізнесів та масштабів використання.

ВИСНОВКИ

Під час виконання бакалаврської кваліфікаційної роботи досягнуто мету, розроблено мобільний застосунок для управління логістикою малого підприємства з використанням сучасних хмарних технологій.

Виконано аналіз поточного стану логістичних процесів малих підприємств та виявлено ключові проблеми, такі як відсутність зручних мобільних рішень, недостатня автоматизація та висока залежність від ручного управління.

Запропоновано архітектуру мобільного застосунку на базі Android з використанням Firebase як хмарної бази даних та Google Maps API для побудови маршрутів і візуалізації точок доставки.

Реалізовано повний набір основних функцій: реєстрація користувачів із ролями, підтвердження акаунтів, створення та повторне використання маршрутів, динамічне оновлення геолокації водія та відображення прогресу доставки.

Проведено тестування застосунку на реальному пристрої та емуляторі, що підтвердило його стабільну роботу, високу продуктивність та готовність до практичного використання.

Розроблений застосунок має гнучку структуру, що надає змогу масштабувати її для більшої кількості користувачів, розширити набір ролей та реалізувати нові функції.

Отже, виконана робота підтверджує очікування та доцільність впровадження мобільного застосунку як ефективного засобу підтримки логістичних процесів, з можливістю подальшого розвитку відповідно до зростаючих потреб бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Google Maps Platform? : вебсайт. URL: https://developers.google.com/maps/faq#whatis (дата звернення: 21.03.2025)

2. Google Maps: вебсайт: вебсайт. URL: https://www.google.com/maps/ (дата звернення: 21.03.2025)

3. OpenStreetMap: вебсайт. URL: https://www.openstreetmap.org/#map=14/49.22446/28.44669 (дата звернення: 23.03.2025)

4. Organizing the mobile workforce optimoroute: вебсайт. URL: https://optimoroute.com/ (дата звернення: 24.03.2025)

5. Android : вебсайт. URL: https://www.android.com/ (дата звернення: 25.03.2025)

6. Кодування додатків для Android (Coding Android Apps) Margaret Kozak Polk 2024 CRC Press (дата звернення: 25.03.2025)

7. Як підтримувати належну роботу додатків і пристроїв за допомогою сервісів Google Play : вебсайт. URL: ttps://support.google.com/googleplay/answer/9037938?hl=uk (дата звернення: 26.03.2025)

8. Як працює Android. Введення для Java-розробників: вебсайт. URL: https://javarush.com/ua/groups/posts/uk.481.jak-pracju-android-vvedennja-dlja-java-rozrobnikv

9. Kotlin Programming Language: вебсайт. URL: https://kotlinlang.org/

10. Xamarin: вебсайт. URL: https://dotnet.microsoft.com/en-us/apps/xamarin

11. Firebase | Google's Mobile and Web App Development Platform : вебсайт. URL: https://firebase.google.com/ (дата звернення: 29.03.2025)

12. Google Maps Platform : вебсайт. URL: https://developers.google.com/maps (дата звернення: 01.04.2025)

13. Android Studio : вебсайт. URL: https://developer.android.com/studio (дата звернення: 04.04.2025)

14. Що таке MVC? Переваги та недоліки MVC : вебсайт. URL: https://alexhost.com/uk/faq/shho-take-mvc-perevagy-ta-nedoliky-mvc (дата звернення: 07.04.2025)

15. Firebase Realtime Database – Google : вебсайт. URL: https://firebase.google.com/docs/database (дата звернення: 07.04.2025)

16. Working with JSON : вебсайт. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Scripting/JSON (дата звернення: 08.04.2025)

17. Firebase Authentication : вебсайт. URL: https://firebase.google.com/docs/auth

18. Xml | API reference: вебсайт. URL: https://developer.android.com/reference/android/util/Xml (дата звернення: 08.04.2025)

19. Android API reference: вебсайт. URL: https://developer.android.com/reference. (дата звернення: 08.04.2025)

20. Create and manage virtual devices | Android Studio : вебсайт. URL: https://developer.android.com/studio/run/managing-avds

21. IntelliJ IDEA – the IDE for Pro Java and Kotlin Development : вебсайт. URL: https://www.jetbrains.com/idea/ (дата звернення: 12.04.2025)

22. Add Firebase to your Android project : вебсайт. URL: https://firebase.google.com/docs/android/setup (дата звернення: 12.04.2025)

23. Maps SDK for Android overview : вебсайт. URL: https://developers.google.com/maps/documentation/android-sdk/overview (дата звернення: 18.04.2025)

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

проф., д.т.н.. Азаров О.Д.

21 березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«Мобільний застосунок для управління логістикою малих підприємств»

08-54.БКР.006.00.000 ТЗ

Науковий керівник: к.т.н. доцент каф. ОТ

_____Кадук О.В.

Студент групи 1КІ-23мс

_____Гринчак В.І.

м. Вінниця — 2025

1 Підставою для виконання бакалаврської кваліфікаційної роботи (БКР)

1.1 У сучасних умовах цифровізації бізнесу логістика стає одним із найважливіших елементів ефективного функціонування підприємств. Проте більшість таких рішень орієнтовані на середній і великий бізнес та є недоступними для впровадження малими підприємствами через високу вартість або складність використання.

1.2 Наказ про затвердження теми кваліфікаційної роботи.

2 Мета БКР і призначення розробки

2.1 Мета роботи — створити мобільний застосунок для управління логістикою малих підприємств з використанням хмарних технологій.

2.2 Призначення розробки — управління логістикою малих підприємств.

3 Вихідні дані для виконання БКР

3.1 Проведення аналізу існуючих систем та технологій управління логістикою малих підприємств;

3.2 Розробка структури та функціональної схеми мобільного застосунку для управління логістикою малих підприємств;

3.3 Розробка бази даних та хмарного середовища;

3.4 Тестування мобільного застосунку.

4 Вимоги до виконання БКР

Головна вимога – створити мобільний застосунок, який матиме практичне застосування в управлінні логістикою малих підприємств.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи БКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Огляд і аналіз сучасного стану та проблем у сфері логістики малих підприємств	21.03.25	30.03.25	Аналітичний огляд літературних джерел
2	Розробка архітектури та функціональних рішень	21.03.25	30.03.25	Розділ 2
3	Програмна реалізація мобільного застосунку	31.03.25	13.04.25	Розділ 3
4	Тестування мобільного застосунку	14.04.25	27.04.25	Розділ 4
5	Апробація та впровадження результатів дослідження	28.04.25	11.05.2025	Тези доповідей
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	28.04.25	11.05.2025	пояснювальна записка, графічний матеріал і/або презентація

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, графічні і ілюстративні матеріали, протокол попереднього захисту БКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до БКР українською та іноземною мовами, довідка про відповідність оформлення БКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів графічної та розрахункової документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлюванні БКР використовуються:

— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 123 «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія») — кафедра обчислювальної техніки, ВНТУ 2023.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

ДОДАТОК Б

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА

НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи:

Мобільний застосунок для управління логістикою малих підприємству

Тип роботи: Бакалаврська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності StrikePlagiarism

Оригінальність 98,5% Схожість 1,5%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою StrikePlagiarism щодо роботи.

Автор роботи _____ Гринчак В.І.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Кадук О. В.
(підпис) (прізвище, ініціали)

ДОДАТОК В

ІЛЮСТРАТИВНА ЧАСТИНА

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ УПРАВЛІННЯ ЛОГІСТИКОЮ МАЛИХ ПІДПРИЄМСТВ

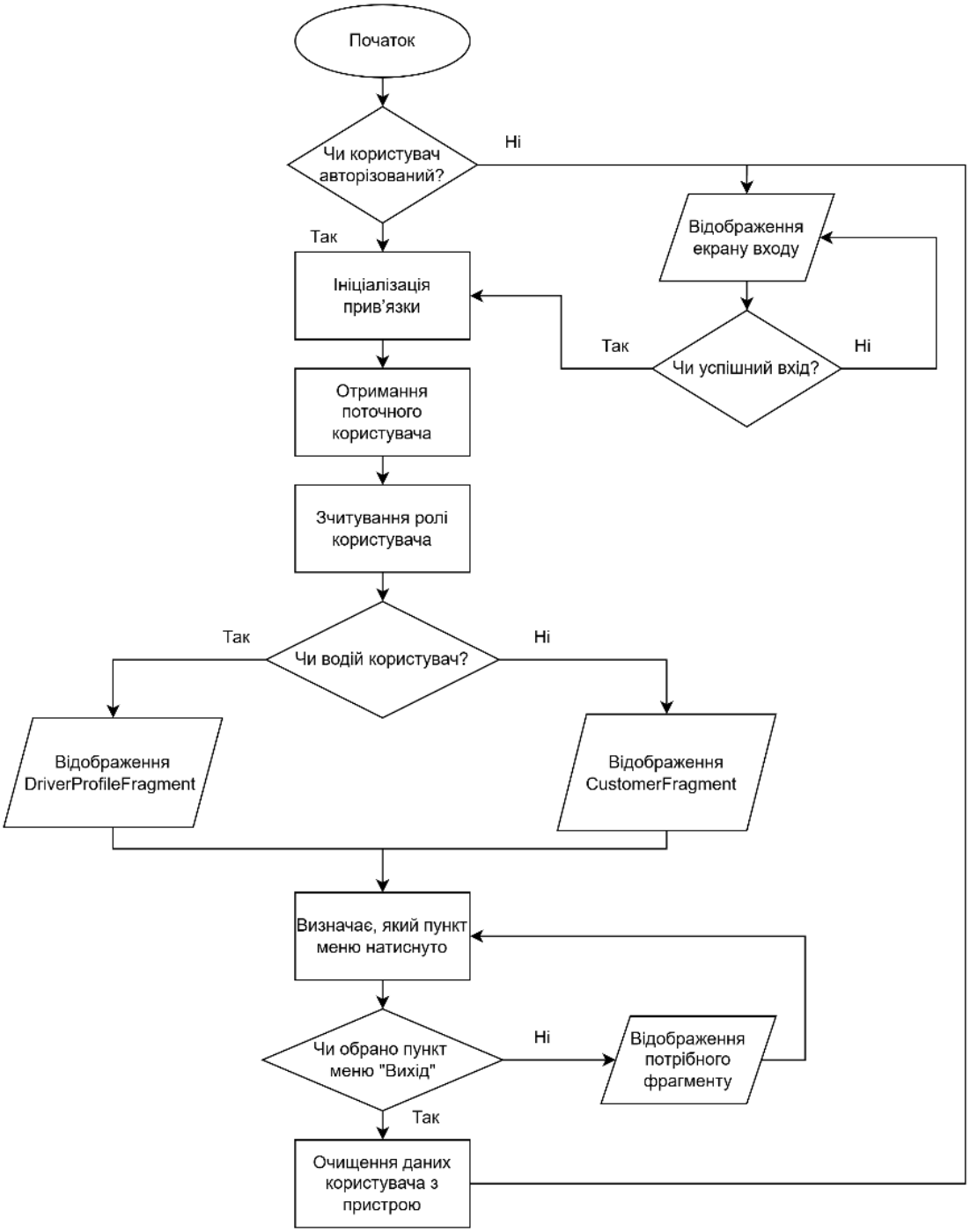

Рисунок В.1 — Алгоритм дій модуля MainActivity

Рисунок В.2 — Схема структури зберігання даних у хмарі

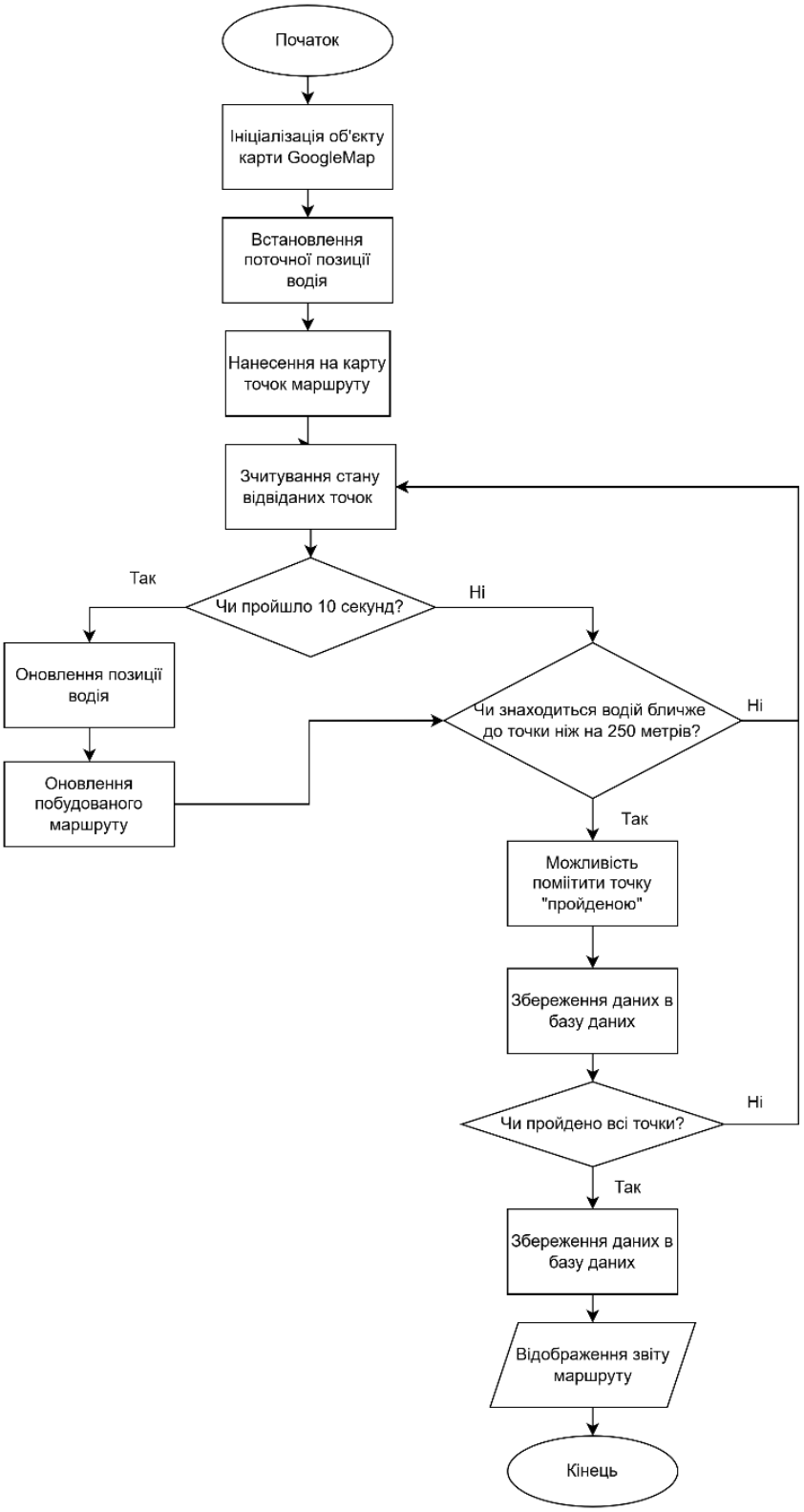

Рисунок В.3 — Алгоритм дій модуля NavigationFragment

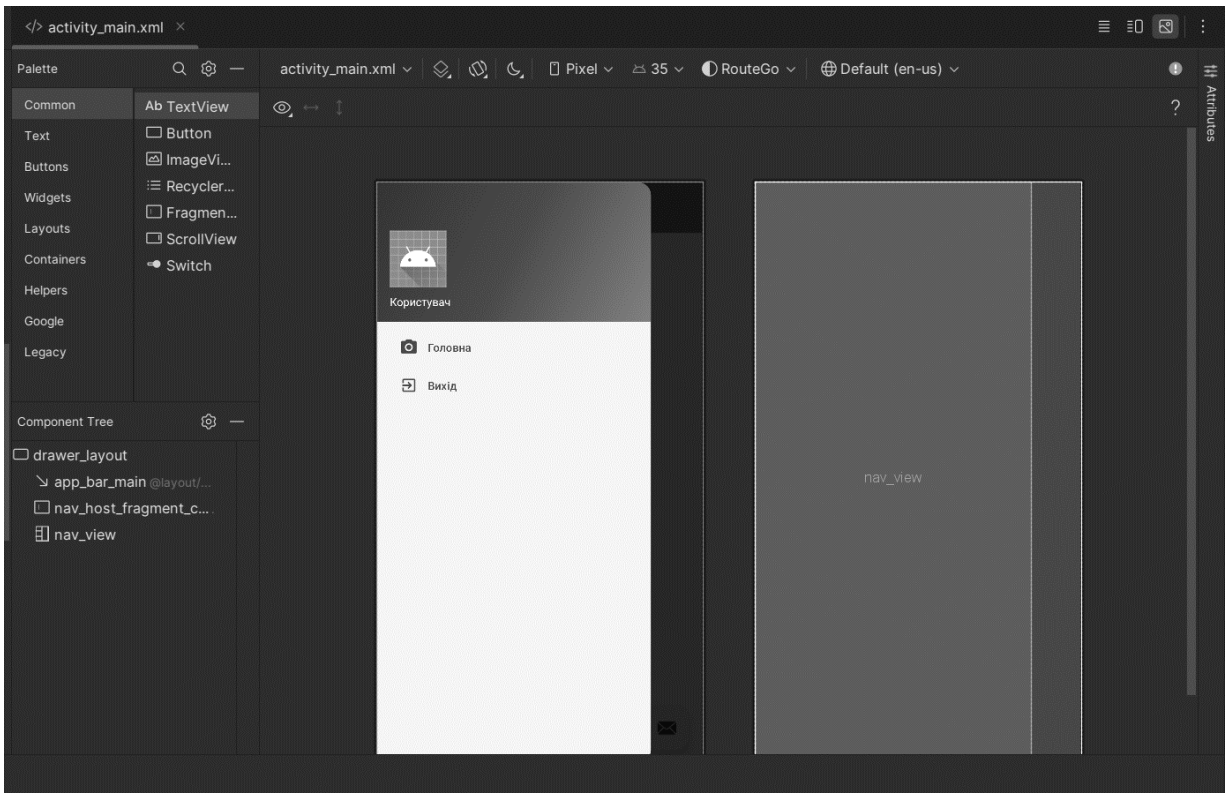

Рисунок В.4 — Дизайн інтерфейсу activity_main.xml

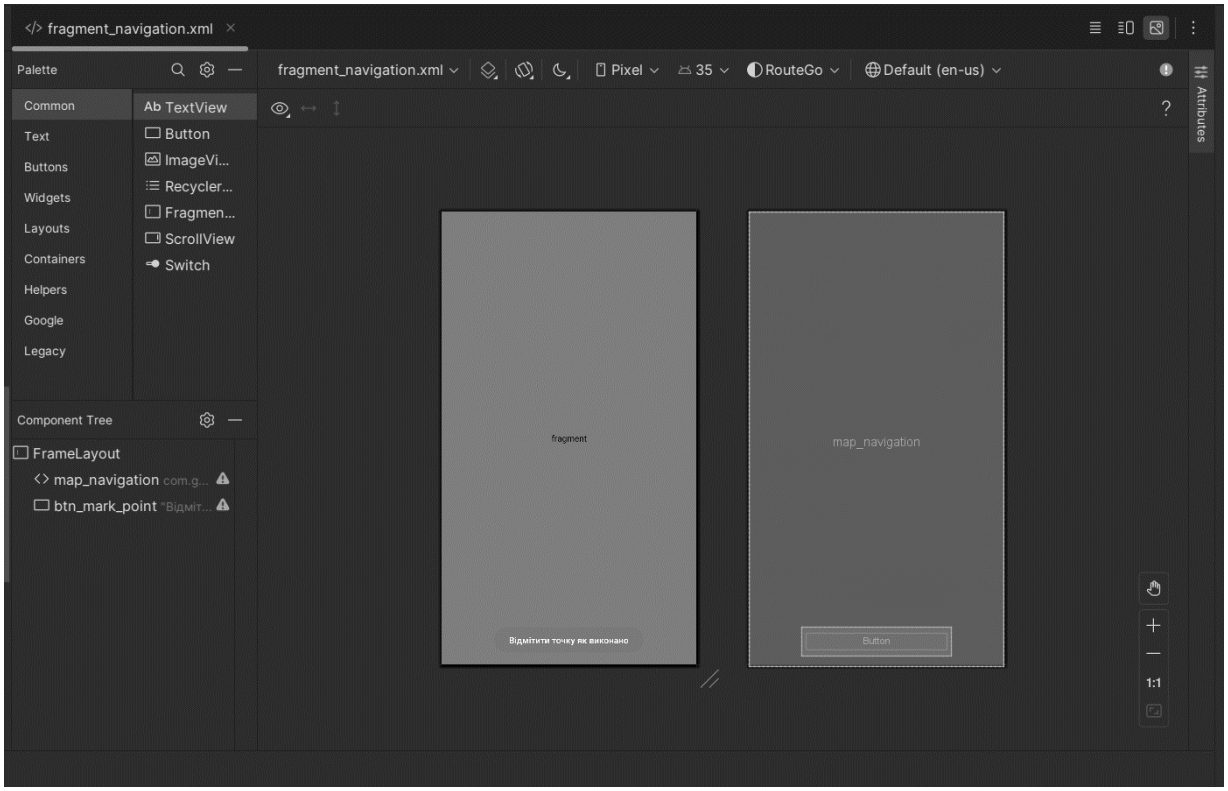

Рисунок В.5 — Дизайн інтерфейсу fragment_navigation.xml

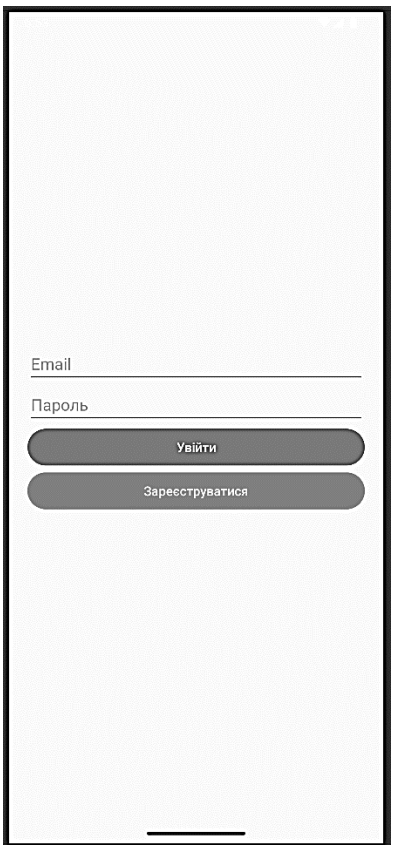

Рисунок В.6 — Графічний інтерфейс LoginActivity

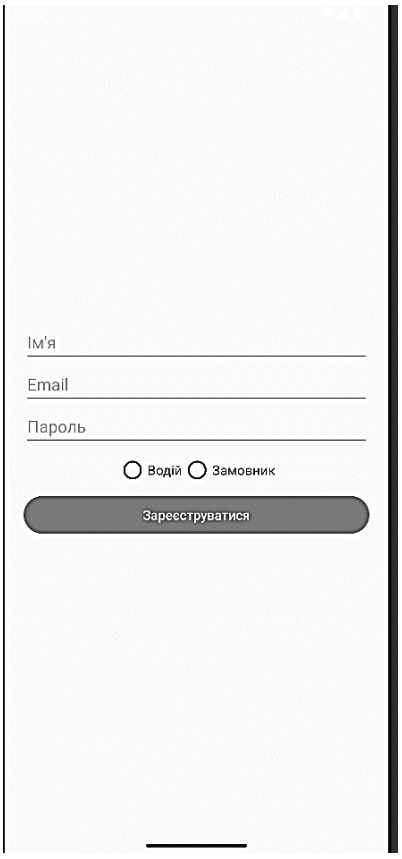

Рисунок В.7 — Графічний інтерфейс RegisterActivity

Рисунок В.8 — Графічний інтерфейс SupplierProfileFragment

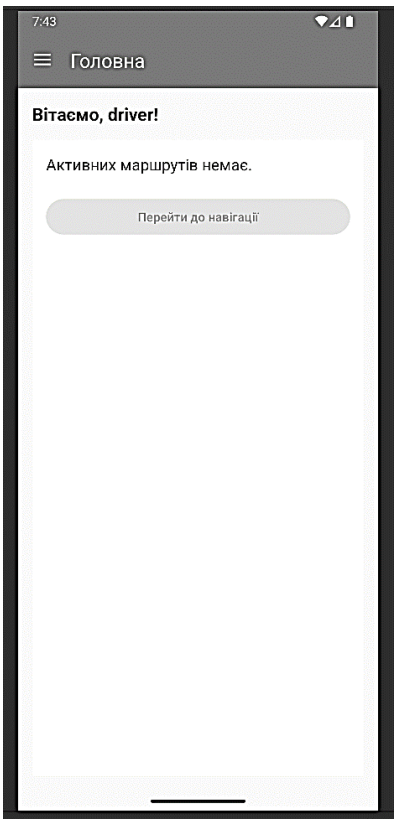

Рисунок В.9 — Графічний інтерфейс DriverRoutesFragment

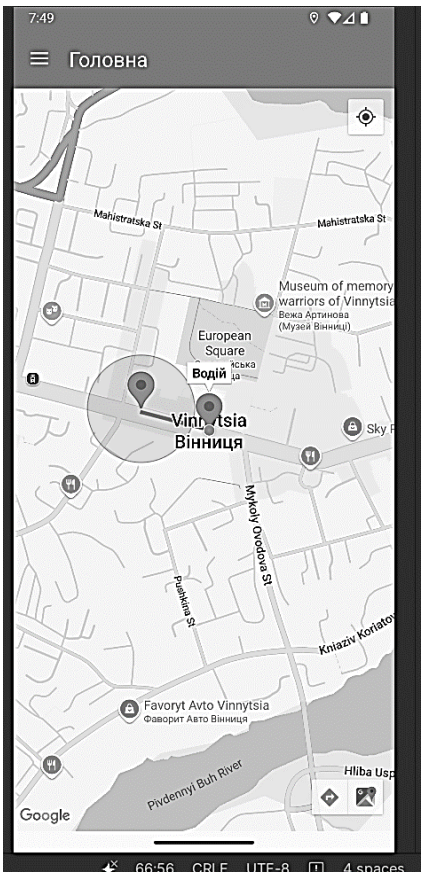

Рисунок В.10 — Графічний інтерфейс NavigationFragment

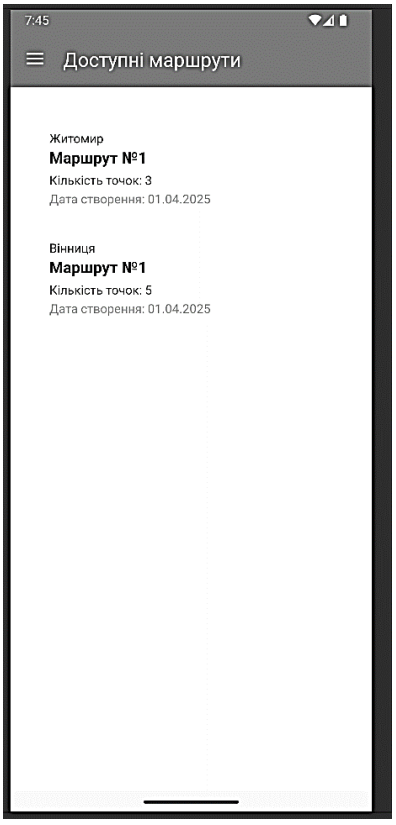

Рисунок В.11 — Графічний інтерфейс «Доступних маршрутуів»

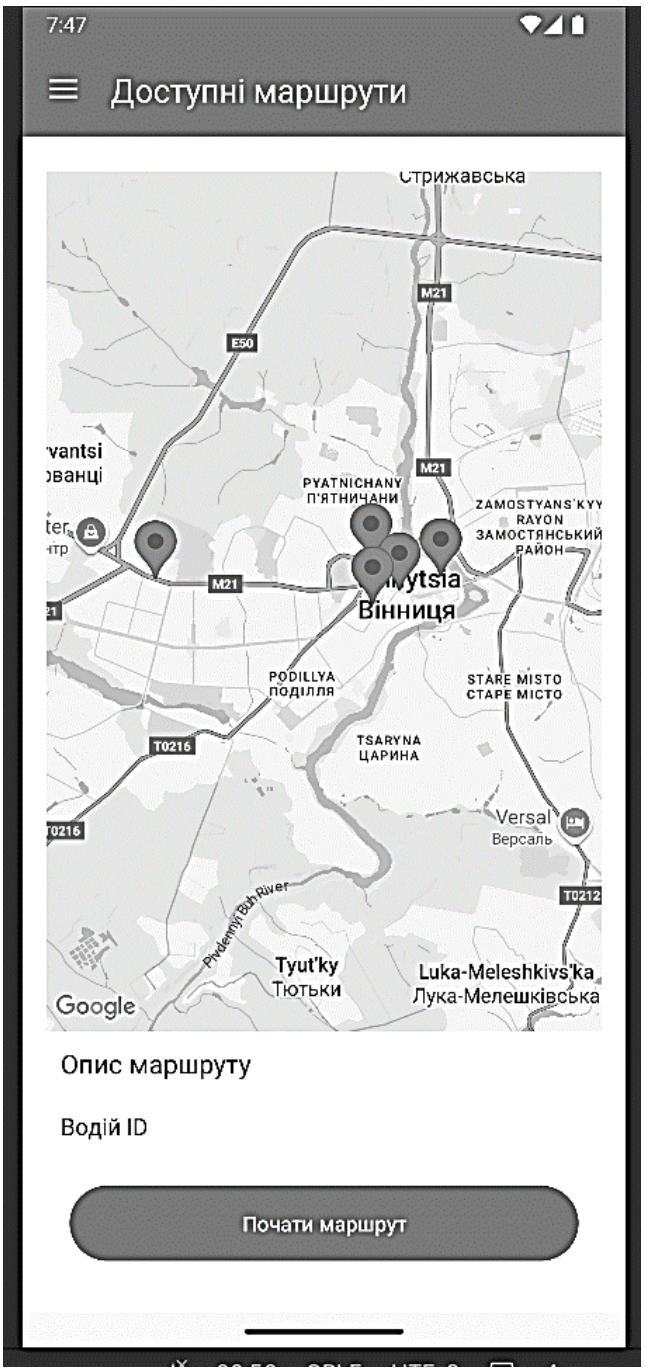

Рисунок В.12 — Графічний інтерфейс «Перегляду маршруту»

ДОДАТОК Д

Лістінг програмного коду файлу NavigationFragment.java

```
package com.example.routego.Driver;

import android.Manifest;
import android.content.pm.PackageManager;
import android.location.Location;
import android.os.Bundle;
import android.util.Log;
import android.view.*;
import android.widget.Button;
import android.widget.Toast;
import androidx.annotation.*;
import androidx.core.app.ActivityCompat;
import androidx.fragment.app.Fragment;

import com.example.routego.R;
import com.google.android.gms.location.*;
import com.google.android.gms.maps.*;
import com.google.android.gms.maps.model.*;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.*;

import java.util.*;

public class NavigationFragment extends Fragment implements OnMapReadyCallback {

    private String driverId = FirebaseAuth.getInstance().getCurrentUser().getUid();

    private GoogleMap mMap;
    private String routeId;
    private DatabaseReference routeRef;

    private FusedLocationProviderClient fusedLocationClient;
    private Location currentLocation;

    private List<LatLng> routePoints = new ArrayList<>();
    private List<String> pointDescriptions = new ArrayList<>();

    //Прогрес
    private List<Boolean> visitedPoints = new ArrayList<>();

    private Marker driverMarker;
```

```
    //навігація
    private LocationCallback locationCallback;

    private RouteManager routeManager;

    private long lastRouteUpdateTime = 0;
    private static final long ROUTE_UPDATE_INTERVAL = 10_000; // 10 секунд

    private Button btnMarkPoint;


    private int currentPointIndex = 0; // точка маршруту, куди рухається водій


    public NavigationFragment() {}

    private boolean allPointsVisited() {
        for (Boolean visited : visitedPoints) {
            if (!visited) return false;
        }
        return true;
    }


    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        requestLocationPermission();

        btnMarkPoint = view.findViewById(R.id.btn_mark_point);
        btnMarkPoint.setVisibility(View.GONE);
        btnMarkPoint.setOnClickListener(v -> {
            if (currentLocation == null || routePoints.isEmpty()) {
                Toast.makeText(getContext(), "Локація або маршрут недоступні",
Toast.LENGTH_SHORT).show();
                return;
            }

            // Знаходимо найближчу непройдену точку
            double minDistance = Double.MAX_VALUE;
            int nearestUnvisitedIndex = -1;
```

```
        for (int i = 0; i < routePoints.size(); i++) {
            if (visitedPoints.get(i)) continue;

            LatLng point = routePoints.get(i);
            float[] results = new float[1];
            Location.distanceBetween(
                    currentLocation.getLatitude(), currentLocation.getLongitude(),
                    point.latitude, point.longitude,
                    results
            );

            if (results[0] < minDistance) {
                minDistance = results[0];
                nearestUnvisitedIndex = i;
            }
        }

        if (nearestUnvisitedIndex == -1) {
            Toast.makeText(getContext(), "Усі точки вже пройдено",
Toast.LENGTH_SHORT).show();
            return;
        }

        if (minDistance <= 250.0f) {
            int indexToMark = nearestUnvisitedIndex;

routeRef.child("points").child(String.valueOf(indexToMark)).child("visited").setValue(t
rue)
                    .addOnSuccessListener(aVoid -> {
                        visitedPoints.set(indexToMark, true);
                        LatLng completedPoint = routePoints.get(indexToMark);
                        routeManager.drawTemporaryCircle(completedPoint, 3000);


                        // ВИКЛИК ОНОВЛЕННЯ МАРКЕРІВ
                        refreshMapMarkers();

                        if (allPointsVisited()) {
                            updateRouteStatus("completed");
                            routeRef.child("complete_seen").setValue(false);
                            routeRef.child("endedAt").setValue(ServerValue.TIMESTAMP);

                            RouteSummaryFragment summaryFragment =
RouteSummaryFragment.newInstance(routeId, "driver");
                            requireActivity().getSupportFragmentManager()
```

```
                        .beginTransaction()
                        .replace(R.id.nav_host_fragment_content_main,
summaryFragment)
                        .commit();
                } else {
                    Toast.makeText(getContext(), "Точку пройдено! Наступна
активна.", Toast.LENGTH_SHORT).show();
                    updateRouteToNextPoint();
                }
            })
            .addOnFailureListener(e ->
                Toast.makeText(getContext(), "Помилка оновлення точки",
Toast.LENGTH_SHORT).show());
        } else {
            Toast.makeText(getContext(), "Ви ще не біля жодної цільової точки",
Toast.LENGTH_SHORT).show();
        }
    });

  }


  public static NavigationFragment newInstance(String routeId) {
    NavigationFragment fragment = new NavigationFragment();
    Bundle args = new Bundle();
    args.putString("routeId", routeId);
    fragment.setArguments(args);
    return fragment;
  }

  @Override
  public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_navigation, container, false);

    if (getArguments() != null) {
      routeId = getArguments().getString("routeId");
    }

    fusedLocationClient                                                    =
LocationServices.getFusedLocationProviderClient(requireContext());

    SupportMapFragment mapFragment = (SupportMapFragment)
getChildFragmentManager().findFragmentById(R.id.map_navigation);
    if (mapFragment == null) {
```

```
            mapFragment = SupportMapFragment.newInstance();
            getChildFragmentManager().beginTransaction().replace(R.id.map_navigation,
mapFragment).commit();
            getChildFragmentManager().executePendingTransactions();
        }

        mapFragment.getMapAsync(this);
        return view;
    }


    private static final int LOCATION_PERMISSION_REQUEST_CODE = 1001;

    private void requestLocationPermission() {
        if (ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
                ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

            requestPermissions(
                    new String[]{
                            Manifest.permission.ACCESS_FINE_LOCATION,
                            Manifest.permission.ACCESS_COARSE_LOCATION
                    },
                    LOCATION_PERMISSION_REQUEST_CODE
            );
        } else {
            enableUserLocation();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);

        if (requestCode == LOCATION_PERMISSION_REQUEST_CODE) {
            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                enableUserLocation();
            } else {
                Toast.makeText(getContext(), "Доступ до геолокації відхилено",
Toast.LENGTH_SHORT).show();
```

```
        }
    }
}

private void enableUserLocation() {
    if (mMap == null) return;

    if (checkLocationPermission()) {
        try {
            mMap.setMyLocationEnabled(true);
        } catch (SecurityException e) {
            e.printStackTrace();
            Toast.makeText(getContext(), "Доступ до геолокації неможливий",
Toast.LENGTH_SHORT).show();
        }
    }
}


@Override
public void onMapReady(@NonNull GoogleMap googleMap) {
    mMap = googleMap;
    routeManager = new RouteManager(mMap,
"AIzaSyAFLceKQsEwQ7EhHB9gPHnuDlLfdcsj-Wg");


    if (ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {
        requestLocationPermission(); // <-- тут
        return;
    }

    enableLocationFeatures();
}

private void refreshMapMarkers() {
    mMap.clear();
    boolean hasUnvisited = false;

    for (int i = 0; i < routePoints.size(); i++) {
        LatLng point = routePoints.get(i);
```

```
            boolean visited = visitedPoints.get(i);
            String description = pointDescriptions.size() > i ? pointDescriptions.get(i) : "";

            String title = " 📍 Маршрутна точка";
            if (visited) {
                title = "☑ Пройдена точка";
            } else if (!hasUnvisited) {
                title = " 🎯 Ціль";
                hasUnvisited = true;
            }

            mMap.addMarker(new MarkerOptions()
                    .position(point)
                    .title(title)
                    .snippet(!description.isEmpty() ? description : null)
                    .icon(BitmapDescriptorFactory.defaultMarker(
                            visited
                                    ? BitmapDescriptorFactory.HUE_CYAN
                                    : (title.equals(" 🎯 Ціль") ? BitmapDescriptorFactory.HUE_ORANGE : BitmapDescriptorFactory.HUE_GREEN)
                    )));
        }
    }

    private void enableLocationFeatures() {
        try {
            mMap.setMyLocationEnabled(true);

            fusedLocationClient.getLastLocation().addOnSuccessListener(location -> {
                if (location != null) {
                    currentLocation = location;
                    LatLng currentLatLng = new LatLng(location.getLatitude(), location.getLongitude());

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(currentLatLng, 15));
                    loadRouteFromFirebase();
                }
            });
        } catch (SecurityException e) {
            e.printStackTrace();
            Toast.makeText(getContext(), "Помилка доступу до геолокації", Toast.LENGTH_SHORT).show();
        }
    }
```

```
    private void loadRouteFromFirebase() {
        routeRef = FirebaseDatabase.getInstance().getReference("Routes").child(routeId);
        routeRef.child("startedAt").get().addOnSuccessListener(snapshot -> {
            if (!snapshot.exists()) {
                routeRef.child("startedAt").setValue(ServerValue.TIMESTAMP);
            }
        });

        routeRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                routePoints.clear();
                visitedPoints.clear();
                pointDescriptions.clear();

                boolean hasUnvisited = false;
                int index = 0;

                for (DataSnapshot pointSnapshot : snapshot.child("points").getChildren()) {
                    Double lat = pointSnapshot.child("latitude").getValue(Double.class);
                    Double lng = pointSnapshot.child("longitude").getValue(Double.class);
                    Boolean visited = pointSnapshot.child("visited").getValue(Boolean.class);
                    String description =
pointSnapshot.child("description").getValue(String.class);

                    if (lat != null && lng != null) {
                        LatLng point = new LatLng(lat, lng);
                        routePoints.add(point);
                        visitedPoints.add(Boolean.TRUE.equals(visited));
                        pointDescriptions.add(description != null ? description : "");

                        String title = "📍 Маршрутна точка";
                        if (Boolean.TRUE.equals(visited)) {
                            title = "☑️ Пройдена точка";
                        } else if (!hasUnvisited) {
                            title = "🎯 Ціль";
                            hasUnvisited = true;
                        }

                        mMap.addMarker(new MarkerOptions()
                                .position(point)
                                .title(title)
                                .snippet(description != null && !description.isEmpty() ? description
```

```
: null)

.icon(BitmapDescriptorFactory.defaultMarker(Boolean.TRUE.equals(visited)
                        ? BitmapDescriptorFactory.HUE_CYAN
                        : (title.equals("🎯 Ціль") ?
BitmapDescriptorFactory.HUE_ORANGE :
BitmapDescriptorFactory.HUE_GREEN))));
            }

            index++;
        }

        if (currentLocation != null) {
            sortRoutePointsByProximity();
        }

        if (!routePoints.isEmpty()) {
            routeManager.setDeliveryPoints(routePoints);
            updateRouteStatus("active");
            currentPointIndex = 0;
            updateRouteToNextPoint();
            startLocationUpdates();
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Toast.makeText(getContext(), "Помилка завантаження маршруту",
Toast.LENGTH_SHORT).show();
    }
  });

}

  private void sortRoutePointsByProximity() {
    LatLng driverPos = new LatLng(currentLocation.getLatitude(),
currentLocation.getLongitude());

    List<LatLng> sorted = new ArrayList<>();
    List<Boolean> visitedSorted = new ArrayList<>();
    List<Integer> indices = new ArrayList<>();

    List<LatLng> tempPoints = new ArrayList<>(routePoints);
    List<Boolean> tempVisited = new ArrayList<>(visitedPoints);
```

```
        while (!tempPoints.isEmpty()) {
            double minDist = Double.MAX_VALUE;
            int nearestIndex = -1;

            for (int i = 0; i < tempPoints.size(); i++) {
                double dist = distanceBetween(driverPos, tempPoints.get(i));
                if (dist < minDist) {
                    minDist = dist;
                    nearestIndex = i;
                }
            }

            if (nearestIndex != -1) {
                sorted.add(tempPoints.get(nearestIndex));
                visitedSorted.add(tempVisited.get(nearestIndex));
                driverPos = tempPoints.get(nearestIndex);

                tempPoints.remove(nearestIndex);
                tempVisited.remove(nearestIndex);
            }
        }

        routePoints = sorted;
        visitedPoints = visitedSorted;
    }

    private void startLocationUpdates() {
        LocationRequest locationRequest = LocationRequest.create();
        locationRequest.setInterval(4000);
        locationRequest.setFastestInterval(2000);
        locationRequest.setPriority(Priority.PRIORITY_HIGH_ACCURACY);

        // ІНІЦІАЛІЗУЄМО CALLBACK ДО requestLocationUpdates
        locationCallback = new LocationCallback() {
            @Override

            public void onLocationResult(@NonNull LocationResult locationResult) {
                currentLocation = locationResult.getLastLocation();
                if (currentLocation != null) {
                    LatLng driverLatLng = new LatLng(currentLocation.getLatitude(),
currentLocation.getLongitude());
                    updateDriverMarker(driverLatLng);
                    updateDriverLocationInFirebase(currentLocation);

                    // ОНОВЛЕННЯ ШЛЯХУ кожні 10 секунд
```

```
                long currentTime = System.currentTimeMillis();
                if (currentTime - lastRouteUpdateTime > ROUTE_UPDATE_INTERVAL)
{
                    updateRouteToNextPoint(); // оновлюємо лінію
                    lastRouteUpdateTime = currentTime;
                }

                if (currentPointIndex < routePoints.size()) {
                    double distance = distanceBetween(driverLatLng,
routePoints.get(currentPointIndex));
                    Log.d("DISTANCE", "Відстань до точки: " + distance);

                    if (distance <= 100) {
                        btnMarkPoint.setVisibility(View.VISIBLE);
                    } else {
                        btnMarkPoint.setVisibility(View.GONE);
                    }
                } else {
                    btnMarkPoint.setVisibility(View.GONE); // безпечне завершення
                }

                // Якщо дійшов до точки
                if (currentPointIndex < routePoints.size()) {
                    double dist = distanceBetween(driverLatLng,
routePoints.get(currentPointIndex));
                    Log.d("DIST_CHECK", "Відстань до цілі: " + dist + "м");
                }

            }
        }

    };

    if (checkLocationPermission()) {
        try {
            fusedLocationClient.requestLocationUpdates(locationRequest,
locationCallback, null);
        } catch (SecurityException e) {
            e.printStackTrace();
            Toast.makeText(getContext(), "Не вдалося увімкнути оновлення
геолокації", Toast.LENGTH_SHORT).show();
        }
    }
}
```

```
    private boolean checkLocationPermission() {
        return ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED ||
                ActivityCompat.checkSelfPermission(requireContext(),
Manifest.permission.ACCESS_COARSE_LOCATION) ==
PackageManager.PERMISSION_GRANTED;
    }

    private void updateRouteToNextPoint() {
        if (routePoints.isEmpty() || currentLocation == null) return;

        double minDistance = Double.MAX_VALUE;
        int nearestIndex = -1;

        LatLng driverPos = new LatLng(currentLocation.getLatitude(),
currentLocation.getLongitude());

        for (int i = 0; i < routePoints.size(); i++) {
            if (visitedPoints.get(i)) continue; // пропускаємо вже пройдені точки

            LatLng point = routePoints.get(i);
            double dist = distanceBetween(driverPos, point);
            if (dist < minDistance) {
                minDistance = dist;
                nearestIndex = i;
            }
        }

        if (nearestIndex != -1) {
            currentPointIndex = nearestIndex;

            LatLng target = routePoints.get(currentPointIndex);
            Location loc = new Location("");
            loc.setLatitude(currentLocation.getLatitude());
            loc.setLongitude(currentLocation.getLongitude());

            routeManager.updateRoute(loc, target);

            Log.d("NavigationFragment", "Оновлено маршрут до точки " +
currentPointIndex);
        }
    }

    private void updateDriverMarker(LatLng position) {
```

```
        if (driverMarker == null) {
            driverMarker = mMap.addMarker(new MarkerOptions()
                    .position(position)
                    .title("Водій")

.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE
)));
        } else {
            driverMarker.setPosition(position);
        }
    }

    private void updateDriverLocationInFirebase(Location location) {
        DatabaseReference driverLocationRef = FirebaseDatabase.getInstance()
                .getReference("DriversLocation").child(driverId);

        Map<String, Object> locationData = new HashMap<>();
        locationData.put("latitude", location.getLatitude());
        locationData.put("longitude", location.getLongitude());
        locationData.put("timestamp", System.currentTimeMillis());

        driverLocationRef.setValue(locationData);
    }

    private void updateRouteStatus(String status) {
        routeRef.child("status").setValue(status);
    }

    private double distanceBetween(LatLng p1, LatLng p2) {
        float[] results = new float[1];
        Location.distanceBetween(p1.latitude, p1.longitude, p2.latitude, p2.longitude,
results);
        return results[0];
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        if (routeManager != null) routeManager.clear();
        if (fusedLocationClient != null && locationCallback != null) {
            fusedLocationClient.removeLocationUpdates(locationCallback);
        }
    }
}
```