

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОРЕНДИ
АВТОМОБІЛІВ

DEVELOPMENT OF A MOBILE APPLICATION FOR CAR RENTAL

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ЙОЖИКОВ А.А.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЙОЖИКОВА Анатолія Андрійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного застосунку для оренди автомобілів. Development of a Mobile Application for Car Rental

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань роботи. 2. Розробка архітектури програми. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т.В., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання « 15 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 2
5	Розробка програми.	3-4 тижні	Розділ 3
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Анатолій ЙОЖИКОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ФЕДОРОНЧАК
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 93 с., 11 табл., 46 рис., 3 дод., 12 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, ОРЕНДА АВТОМОБІЛЯ, FIRESTORE, JAVA.

Об'єкт дослідження – програмні засоби для оренди автомобілів.

Предмет дослідження – мобільні застосунки для оренди автомобілів.

Мета роботи – створення Android-застосунку для взяття в оренду автомобілів.

Матеріали, методи та технічні засоби: мова програмування Java, шаблон проектування MVVM, сервіси Firestore та Razorpay, мобільний пристрій з операційною системою Android.

Результати. Створено програмний Android-застосунок, що дозволяє користувачам брати автомобілі в оренду.

Висновок. Розроблено програмний Android-застосунок для взяття в оренду автомобілів.

Галузь використання – автопрокатними компаніями для простішого та зручного управління бронюванням та орендою автомобілів своїм клієнтам. Туристами для подорожей у нових місцях.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 93 pages, 11 tables, 46 figures, 3 appendixes, 12 sources.

MOBILE APPLICATION, ANDROID, CAR RENTAL, FIRESTORE, JAVA.

Object of the study is software for car rental.

Subject of the study is mobile applications for car rental.

The purpose of the work is to create an Android application for renting cars.

Materials, methods and tools: Java programming language, MVVM design template, Firestore and Razorpay services, mobile device with Android operating system.

Results. The Android application has been created that allows users to rent cars.

Conclusion. The Android application for renting cars was developed.

The field of use – by car rental companies for easier and more convenient management of car reservations and rentals for their customers. Tourists for traveling in new places.

ЗМІСТ

С.

Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз проблеми та постановка завдань роботи	10
1.1 Аналіз предметної області	10
1.2 Огляд існуючих аналогів	10
1.2.1 Turo	10
1.2.2 Carngo	11
1.2.3 Getaround	13
1.3 Постановка завдання роботи	14
2 Розробка архітектури програми	16
2.1 Вибір мови програмування	16
2.2 Визначення функціональних вимог до застосунку	17
2.3 Вибір бази даних	18
2.4 Вибір сервісу для оплати оренди	20
2.5 Архітектура застосунку	21
3 Основні рішення щодо реалізації компонентів системи	23
3.1 Алгоритм функціонування застосунку	23
3.2 Опис бази даних	24
3.3 Опис структури застосунку	27
3.3.1 Пакет model та опис його класів	28
3.3.2 Пакет viewmodel та опис його класів	30
3.3.3 Пакет view та опис його класів	32
3.4 Опис алгоритмів функціонування застосунку	34
3.4.1 Алгоритм реєстрації	34
3.4.2 Алгоритм авторизації	34
3.4.3 Алгоритм оформлення оренди	35
3.4.4 Алгоритм додавання даних про водійське посвідчення	35
3.4.5 Алгоритм оплати оренди	36

3.4.6 Алгоритм скасування оренди	36
3.4.7 Алгоритм пошуку автомобіля по бренду або моделі.....	37
4 Експлуатація, тестування та експериментальне дослідження програми..	38
4.1 Призначення застосунку та умови його виконання	38
4.2 Виконання роботи з застосунком.....	38
4.3 Тестування застосунку та його результати	52
Висновки	54
Перелік джерел посилань	55
Додаток А Технічне завдання	57
Додаток Б Текст програми	60
Додаток В Слайди презентації.....	85

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CVV – Card Verification Value;

JSON – JavaScript Object Notation;

MVVM – Model View ViewModel;

NoSQL – Not only SQL;

UI – User Interface;

БД – База даних;

Мб – Мегабайт;

ПІБ – Прізвище Ім'я По батькові.

ВСТУП

У сучасному світі використання автомобілів стало золотим стандартом. Статистика переконливо свідчить на користь того, що у світі з кожним роком стає все більше водіїв. Така тенденція не викликає здивування, тому що використання автомобіля – це зручно, надійно, престижно та навіть економічно. Більш того, сьогодні навіть не обов'язково купувати машину та постійно підтримувати її в задовільному робочому стані – оренда автомобіля дає змогу кожному охочому використовувати сучасні автомобілі в будь-який час в багатьох місцях України. Практика показує, що в сучасному світі оренда автомобіля приваблює багатьох людей та компаній. Дуже часто, при плануванні подорожі в інші регіони країни, автолюбители віддають перевагу громадському пасажирському транспорту – люди добираються до пункту призначення потягами, автобусами чи іншими видами транспорту, а вже на місці беруть автомобіль в оренду. Крім того, послуги з оренди транспортних засобів приваблюють користувачів і в інших ситуаціях, наприклад: перед купівлею власного автомобіля, деякі люди приймають рішення їздити на орендованій машині – це дозволяє накопичити кошти на купівлю та не втратити навички водіння [1].

Тому роботу присвячено розробці мобільного застосунку, який дозволить користувачам з легкістю брати в оренду автомобіль на будь-який термін часу.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ РОБОТИ

1.1 Аналіз предметної області

Оренда автомобіля (прокат автомобіля) – здача автомобіля без водія на деякий час (від декількох годин до кількох місяців чи навіть років) [2].

Найчастіше автомобіль в оренду беруть ті, у кого немає особистого транспортного засобу (протягом певного часу або в цілому). Ще одна категорія користувачів – туристи, які хочуть пересуватися по чужому місту з комфортом. Серед клієнтів, які орендують автомобіль, багато тих, хто планує придбати транспортний засіб, але не може визначитися з машиною, на якій варто зупинити вибір. Окрема група клієнтів – корпоративні користувачі. Багато компаній вдаються до послуг автомобіля на прокат на початкових етапах діяльності. Утримувати автопарк на порядок дорожче, ніж користуватися орендованими автомобілями. Відповідно, поки у компанії немає регулярного прибутку, здатного покривати подібні витрати, краще брати авто в оренду [3].

Тому на сьогоднішній день з'являється багато програмних засобів, які дозволяють з легкістю брати в оренду автомобіль. Одними із них є мобільні застосунки. Мобільні застосунки дозволяють користувачам легко знаходити та бронювати автомобілі в будь-який час і в будь-якому місці. Замість того, щоб брати автомобіль в оренду через вебсайти, користувачі можуть використовувати зручні мобільні застосунки на своїх смартфонах. Також вони дозволяють користувачам переглядати широкий вибір автомобілів різних класів і моделей, що дозволяє їм знайти оптимальний варіант для себе.

1.2 Огляд існуючих аналогів

1.2.1 Turo

Turo – це найбільший у світі ринок однорангового обміну автомобілями, де клієнт може забронювати будь-який автомобіль у місцевих

господарів у Великобританії, Канаді, Франції та Австралії [4]. Застосунок надає користувачам такі послуги: реєстрація в систему та редагування свого профілю, пошук автомобіля за різними критеріями, такими як клас автомобіля, бренд, трансмісія та інші. Також застосунок надає можливість користувачам брати в оренду автомобіль на будь-який термін часу. На рис. 1.1 зображено загальний вигляд застосунку.

Переваги:

- цікавий та зручний інтерфейс;
- гнучкість при виборі автомобіля, локації та часу оренди.

Недоліки:

- недоступний в Україні.

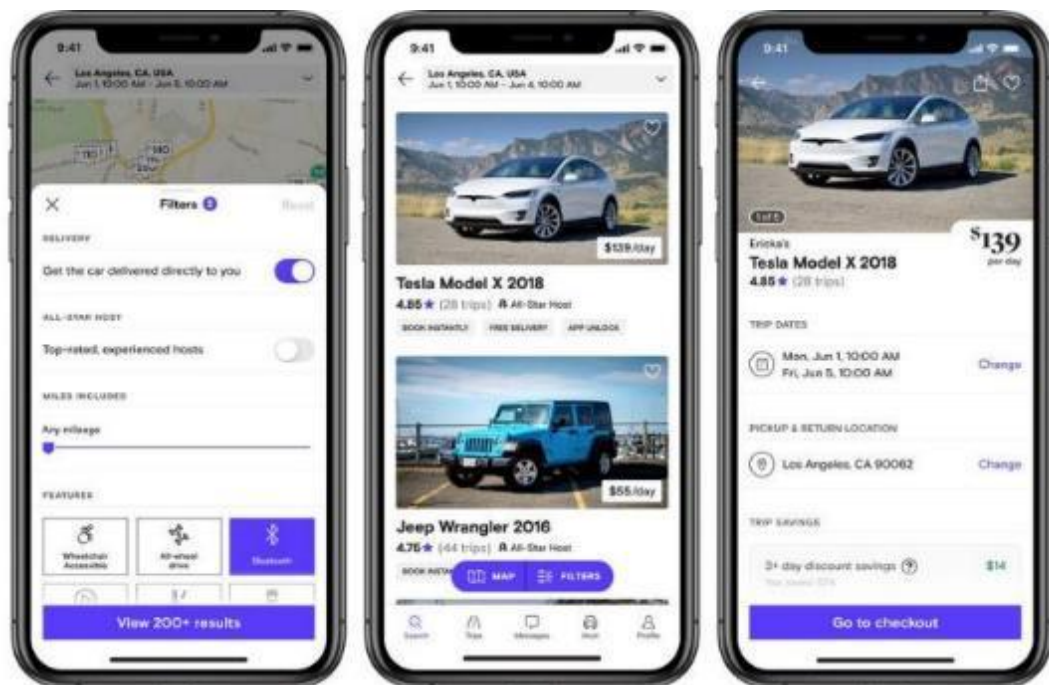


Рисунок 1.1 – Застосунок «Turo» [5]

1.2.2 Carngo

Carngo – це мобільний застосунок для пошуку оренди автомобіля по всьому світу. Цей застосунок містить в собі функцію порівняння цін на оренду

авто від різних компаній. Також доступне меню редагування та скасування бронювання автомобіля [6].

На рис. 1.2 зображено загальний вигляд застосунку.

Переваги:

- зручний інтерфейс;
- широкий вибір автомобілів;
- дозволяє порівнювати ціни на оренду авто від різних компаній.

Недоліки:

- проблеми зі зворотнім зв'язком;
- відсутня можливість брати автомобіль в оренду в будь-який час;
- недоступний в Україні.

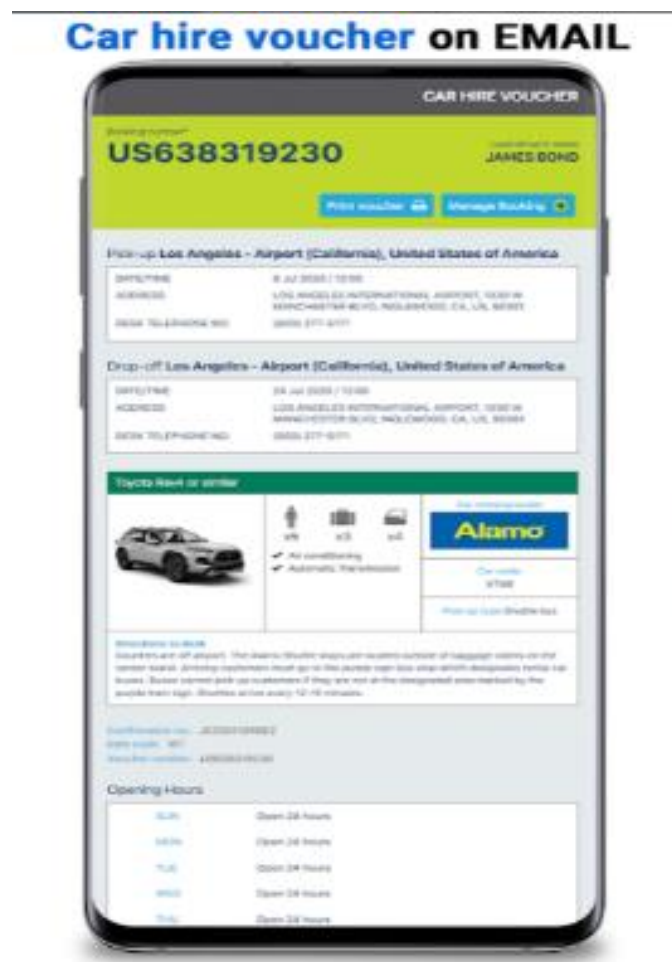


Рисунок 1.2 – Застосунок «Carngo» [6]

1.2.3 Getaround

Getaround – це онлайн-сервіс спільного використання автомобілів або однорангового обміну автомобілями, який з'єднує водіїв, яким потрібно зарезервувати автомобілі, з власниками автомобілів, які діляться своїми автомобілями в обмін на оплату [7]. Цей застосунок дозволяє користувачам створювати облікові записи, вказавши особисті дані та інформацію про водійське посвідчення. Також Getaround надає можливість користувачам шукати автомобіль за маркою, моделлю, класом, ціною та брати в оренду на будь-який термін часу. На рис. 1.3 зображено загальний вигляд застосунку.

Переваги:

- широкий вибір автомобілів;
- гнучкість при виборі автомобіля, локації та часу оренди;
- доступний в Україні.

Недоліки:

- відсутня можливість додавати автомобіль в улюблених.

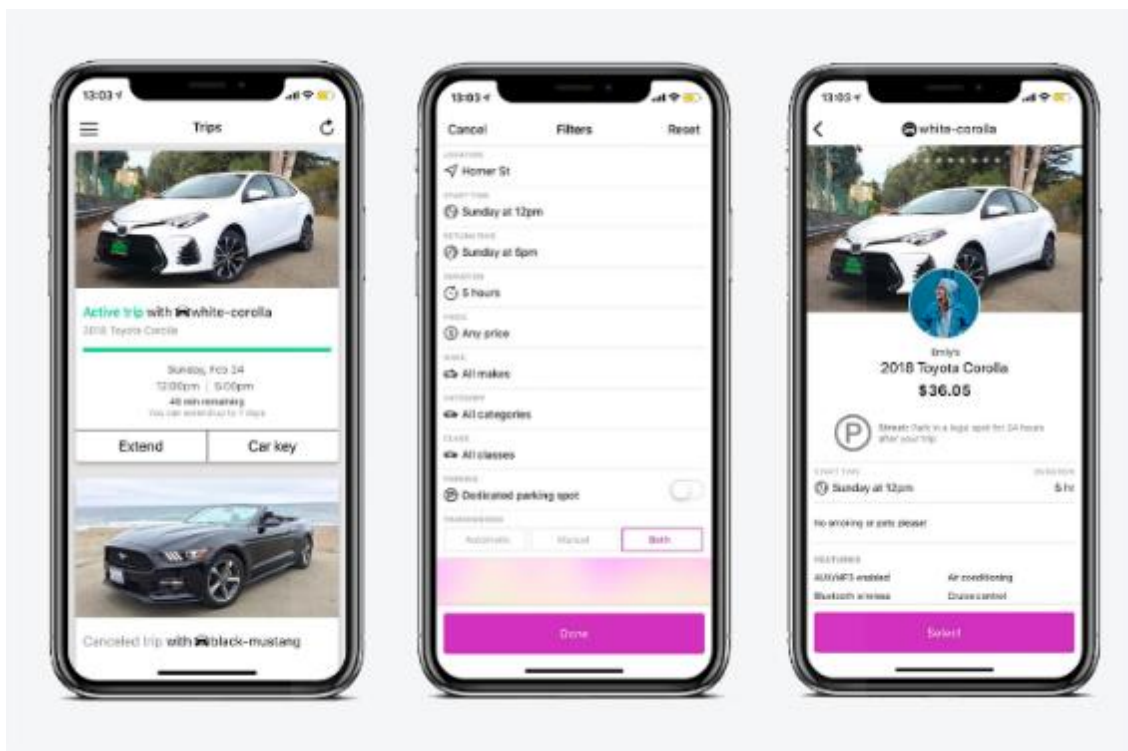


Рисунок 1.3 – Застосунок «Getaround» [8]

Порівняльна характеристика застосунків наведена у табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика застосунків

Критерій порівняння	Turo	Carngo	Getaround
Доступність	Канада, Великобританія	Канада	Європа
Відсутність проблем зі зворотнім зв'язком	+	-	+
Широкий вибір автомобіля	+	+	+
Зручність	+	+	-
Гнучкість при виборі авто, локації та часу оренди	+	-	+

1.3 Постановка завдання роботи

За результатами огляду існуючих аналогів, завдання до роботи – це розробка Android-застосунку для оренди автомобілів.

Застосунок повинен забезпечувати виконання наступного функціонала:

- пошук автомобіля по назві моделі або бренду;
- перегляд детальної інформації про автомобіль;
- оформлення оренди;
- оплата оренди;
- перегляд власних оренд;
- перегляд детальної інформації про оренду;

- скасування оренди протягом двох днів;
- додавання даних про водійське посвідчення.

Що стосується інформації про автомобілі, то вона буде братися з сайту Rental UA та буде додана до бази даних вручну, та використовуватися в застосунку.

Перелік інформації про автомобілі, які будуть взяті з сайту Rental UA:

- назва бренду та моделі автомобіля;
- тип трансмісії та палива;
- фотографії автомобіля;
- ціна;
- об'єм двигуна.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір мови програмування

Java – об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розроблення об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування [9].

Ця класична мова програмування зіграла важливу роль у розробці Android-застосунків. Java була першою офіційною мовою програмування для Android. Це рішення Google ухвалив з метою зробити розробку застосунків на Android більш доступною і поширеною. За допомогою Java було створено величезну кількість застосунків, які досі успішно працюють на мільйонах пристроїв. Одним із великих плюсів Java для Android-розробки є надійність. Застосунки, створені цією мовою, зазвичай працюють дуже стабільно і безпечно. Це, звичайно, важливо для користувачів і забезпечує захист даних. Крім того, у Java величезна спільнота розробників. Це означає, що завжди є доступ до великих ресурсів, підтримки, і безлічі бібліотек і фреймворків, що сильно спрощує розробку [10].

У табл. 2.1 наведено порівняння мови програмування Java з іншими мовами програмування.

Таблиця 2.1 – Порівняння Java з іншими мовами програмування

Критерій порівняння	Java	Kotlin	C++
Продуктивність	Висока	Висока	Середня
Складність розробки	Середня	Середня	Висока
Безпека та надійність	Висока	Середня	Низька
Підтримка спільноти	Висока	Висока	Середня

2.2 Визначення функціональних вимог до застосунку

Функціональні вимоги до застосунку:

- користувач повинен мати можливість створювати облікові записи та авторизуватися;
- користувач повинен мати можливість додавати дані про своє водійське посвідчення;
- користувач повинен мати можливість сплачувати оренду автомобіля через кредитну картку;
- користувач повинен мати можливість переглядати детальну інформацію про конкретний автомобіль;
- користувач повинен мати можливість оформити оренду, вибравши станцію, дату видачі та повернення автомобіля;
- користувач повинен мати можливість шукати автомобіль по назві моделі або бренду;
- користувач повинен мати можливість переглядати свої власні оренди;
- користувач повинен мати можливість переглядати детальну інформацію про оренду;

– користувач повинен мати можливість скасувати оренду протягом двох днів.

На основі вище зазначених функціональних вимог було побудовано діаграму прецедентів. Діаграма прецедентів наведена на рис. 2.1.

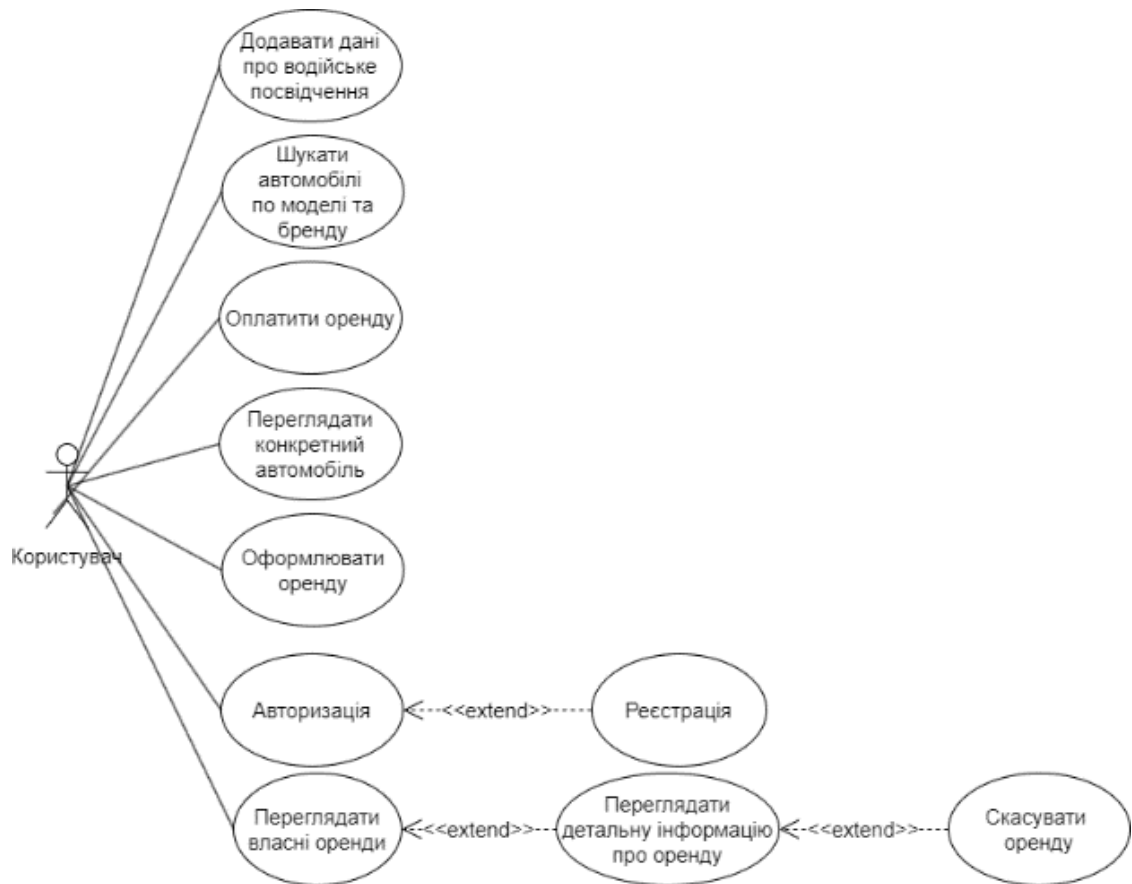


Рисунок 2.1 – Діаграма прецедентів

2.3 Вибір бази даних

Існують безліч типів баз даних, але одними з найпопулярніших є: реляційна та нереляційна база даних.

Реляційна база даних – це тип бази даних, яка організовує дані в таблиці з рядками та стовпцями. Кожна таблиця представляє певну сутність, а стовпці містять атрибути або властивості цієї сутності. Зв'язок між різними таблицями встановлюється за допомогою ключів, які зв'язують їх разом [11].

Нереляційна база даних (також відома як база даних NoSQL) — це тип бази даних, яка не покладається на традиційну реляційну модель на основі таблиць. Нереляційні бази даних часто використовуються, коли дані є неструктурованими або напівструктурованими, тобто вони не вміщуються в рядки та стовпці традиційної таблиці. Це робить їх більш гнучкими, ніж реляційні бази даних, які призначені для роботи з даними, організованими в таблиці [12].

У табл. 2.2 наведена порівняльна характеристика між цими двома типами бази даних.

Таблиця 2.2 – Порівняльна характеристика реляційної та нереляційної бази даних

Опис	Реляційна база даних	Нереляційна база даних
Структура даних	Дані представлені у вигляді таблиць	Дані можуть бути представлені у вигляді документів, графів та тощо
Гнучкість	Менш гнучка	Більш гнучка
Нормалізація	Потребує	Не потребує
Масштабованість	Менш масштабована	Більш масштабована

В якості бази даних було обрано NoSQL через свою масштабованість та гнучкість. Одним із представників, які мають сервіси для побудови бази даних NoSQL є Firebase. Це платформа, розроблена компанією Google, яка надає різноманітні сервіси та інструменти для розробки мобільних застосунків.

Firebase має два сервіси, які пов'язані з побудовою бази даних NoSQL – це Realtime Database та Firestore.

У табл. 2.3 зображена порівняльна характеристика цих двох сервісів.

Таблиця 2.3 – Порівняльна характеристика Realtime Database та Firestore

Опис	Realtime Database	Firestore
Модель даних	Зберігає дані у вигляді дерева JSON	Зберігає дані в документах
Масштабованість	Має певний ліміт в масштабуванні	Автоматично масштабується
Запити	Підтримує глибокі запити зі обмеженими можливостями сортування та фільтрування	Підтримує розширені індексовані запити зі складним сортуванням і фільтруванням

Тому для побудови бази даних NoSQL було обрано сервіс Firestore через свою масштабованість, модель даних та складність запитів.

2.4 Вибір сервісу для оплати оренди

Razorpay – це сервіс, який надає послуги онлайн-платежу. Одним з головних переваг є надання широкого вибору способу оплати, такі як: кредитні або дебетові картки, інтернет-банкінг та інші. Також Razorpay забезпечує високий рівень захисту та конфіденційності даних при транзакції платежу.

У табл. 2.4 наведено порівняння Razorpay з іншими сервісами.

Таблиця 2.4 – Порівняння Razorpay з іншими сервісами способу оплати

Критерій порівняння	Razorpay	PayPal	Stripe
Безпека	Висока	Середня	Середня
Простота налаштування та використання	Висока	Висока	Середня
Широкий вибір способу оплати	+	+	-

2.5 Архітектура застосунку

Для побудови застосунку було обрано шаблон проєктування MVVM, який складається з трьох компонентів: Model, View та ViewModel.

Model відповідає за дані та їхню обробку. По суті Model можна розбити на такі складові: Repository та джерело даних. Джерелом даних може бути БД, API чи інший тип джерела даних звідки вони будуть братися для подальшої обробки. А Repository відповідає за цю саму обробку та відправкою даних до ViewModel.

View відповідає за все те, що пов'язано з UI, а саме: відображенням даних користувачеві, реагуванням на дії користувача, наприклад коли він натиснув на кнопку.

ViewModel є посередником між Model та View, також він відповідає за бізнес-логіку застосунку та відправкою даних всім View, які отримують від нього ці самі дані. Можна сказати, що зв'язок між ViewModel та View один до багатьох і те, що ViewModel немає посилання на конкретний View, як це є в інших шаблонах проєктування таких як MVP та MVC. Тому замість цього View стають спостерігачами, які чекають отримання даних від ViewModel, а в свою чергу ViewModel розсилає всім своїм спостерігачам дані при цьому не знаючи конкретно хто з View спостерігає за ним.

На рис. 2.2 зображено архітектуру застосунку на основі шаблону проєктування MVVM.



Рисунок 2.2 – Шаблон проєктування MVVM

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Алгоритм функціонування застосунку

На рис. 3.1 зображено схему функціонування застосунку.

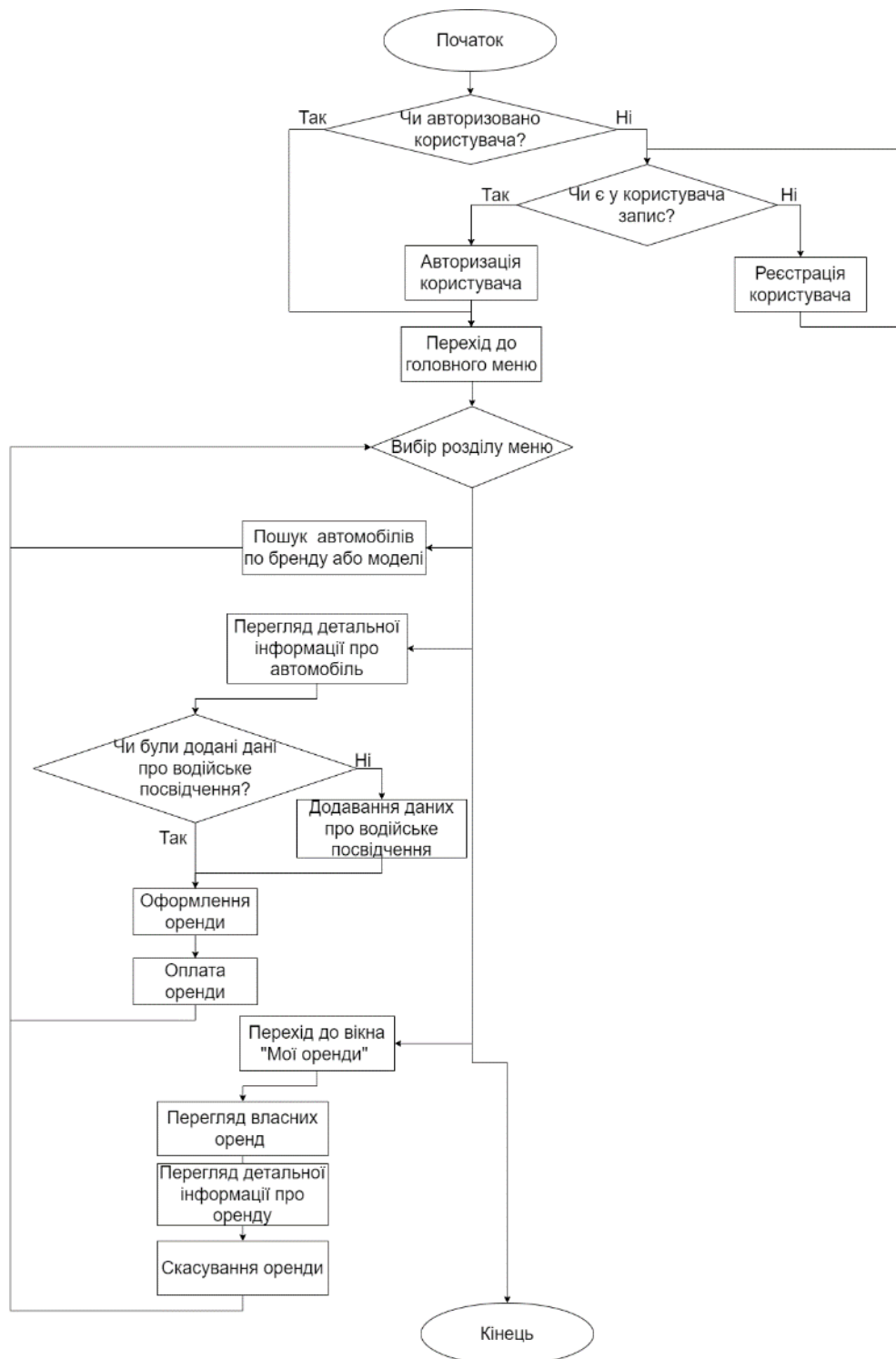


Рисунок 3.1 – Схема функціонування застосунку

3.2 Опис бази даних

У підрозділі 2.3 було обрано базу даних NoSQL та сервіс Firestore для її побудови. База даних складається з таких сутностей:

- users;
- rents;
- driverslicense;
- stations;
- models;
- cars.

Сутність «users» зберігає в собі інформацію про користувача, а саме: його номер, ПІБ, електрона пошта, номер телефону, логін, пароль, номер водійського посвідчення, фото. Поля та їхній тип цієї сутності наведено у табл. 3.1.

Таблиця 3.1 – Сутність «users»

Поле	Тип поля
token	String
fullName	String
email	String
phoneNumber	String
username	String
password	String
driverLicence	String
photo	String

Сутність «rents» зберігає в собі інформацію про оренду, а саме: її номер, номер користувача, номер автомобіля, станція видачі та повернення, дата видачі та повернення автомобіля, загальна сума. Поля та їхній тип цієї сутності наведено у табл. 3.2.

Таблиця 3.2 – Сутність «rents»

Поле	Тип поля
id	String
client	String
car	String
stationIssuing	String
stationReturn	String
issuingDate	String
returnDate	String
totalAmount	int

Сутність «driverslicense» зберігає в собі інформацію про водійське посвідчення користувача, а саме: номер посвідчення, дата видачі та кінцевий термін придатності. Поля та їхній тип цієї сутності наведено у табл. 3.3.

Таблиця 3.3 – Сутність «driverslicense»

Поле	Тип поля
id	String
dateIssuing	String
validDate	String

Сутність «stations» зберігає в собі інформацію про станцію, де користувач буде забирати або повертати автомобіль, а саме: назва станції та її адреса. Поля та їхній тип цієї сутності наведено у табл. 3.4.

Таблиця 3.4 – Сутність «stations»

Поле	Тип поля
name	String
address	String

Сутність «models» зберігає в собі інформацію про модель автомобіля, яку користувач бере в оренду, а саме: назва моделі та виробник. Поля та їхній тип цієї сутності наведено у табл. 3.5.

Таблиця 3.5 – Сутність «models»

Поле	Тип поля
name	String
producer	String

Сутність «cars» зберігає в собі інформацію про автомобіль, який користувач бере в оренду, а саме: номер автомобіля, модель, тип трансмісії, тип палива, ціна, список фотографій автомобіля, наявність дитячого крісла та об'єм двигуна. Поля та їхній тип цієї сутності наведено у табл. 3.6.

Таблиця 3.6 – Сутність «cars»

Поле	Тип поля
ID	String
model	String
transmission	String
typeOfFuel	String
price	int
photo	List<String>
childrenChair	String
engineVolume	String

На рис. 3.2 зображено схему бази даних та зв'язки між сутностями.

Зв'язок між сутностями «users» та «driverslicense» один до одного тому, що користувач може мати лише одне водійське посвідчення, а водійське посвідчення може належати лише одному користувачеві.

Зв'язок між сутностями «users» та «rents» один до багатьох тому, що в одного користувача може бути декілька оренд, а оренда може належати лише одному користувачеві.

Зв'язок між сутностями «stations» та «rents» один до багатьох тому, що одна станція може бути в декількох оренд, а оренда може належати лише одній станції.

Зв'язок між сутностями «cars» та «rents» один до багатьох тому, що один автомобіль може бути в декількох оренд, а оренда може належати лише одному автомобілю.

Зв'язок між сутностями «models» та «cars» один до багатьох тому, що в одній моделі може бути декілька автомобілів, а автомобіль може належати лише одній моделі.

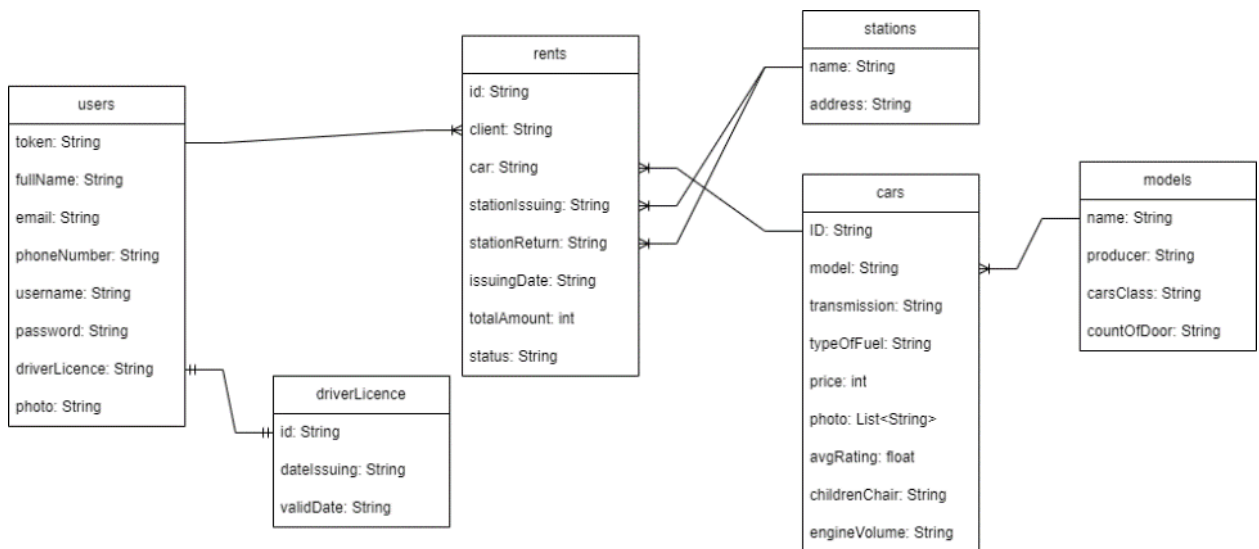


Рисунок 3.2 – Схема бази даних та зв'язки між сутностями

3.3 Опис структури застосунку

У підрозділі 2.5 для побудови застосунку було обрано шаблон проєктування MVVM. На рис. 3.3 зображено структуру проєкту на основі шаблону проєктування MVVM.

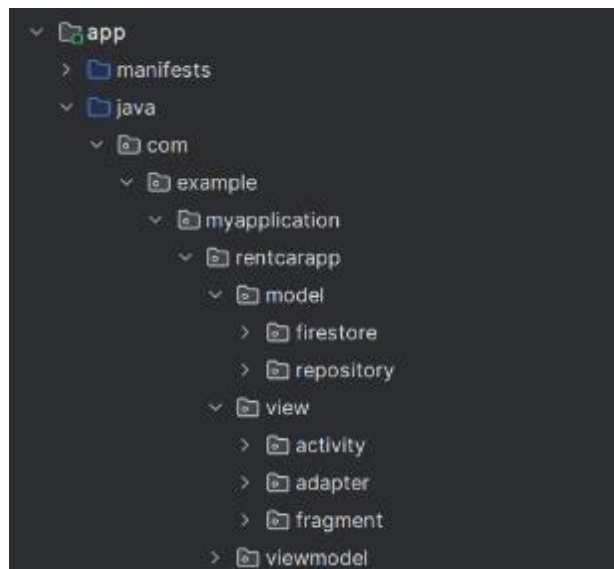


Рисунок 3.3 – Структура проєкту

3.3.1 Пакет model та опис його класів

В пакеті model (рис. 3.4) містяться класи, які відповідають за дані та їхню обробку.

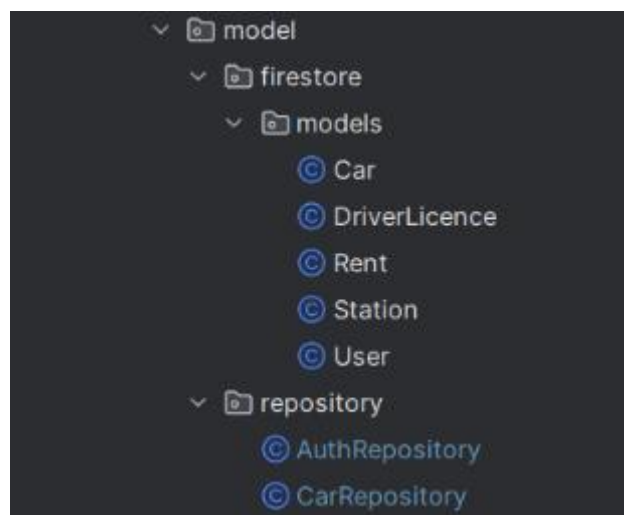


Рисунок 3.4 – Пакет model

Клас Car – це клас, який є моделлю сутності «cars».

Клас DriverLicence – це клас, який є моделлю сутності «driverslicense».

Клас Rent – це клас, який є моделлю сутності «rents».

Клас Station – це клас, який є моделлю сутності «stations».

Клас User – це клас, який є моделлю сутності «users».

Клас AuthRepository – це клас, який відповідає за отримання даних від Firestore. Також відповідає за обробку даних та відправкою даних до AuthViewModel. Основні його методи:

- `public LiveData<User> getUserByUsername(String username)` – це метод, який відповідає за отримання даних про користувача по його логіну з Firestore;

- `public LiveData<User> getUserByUsernameAndPassword(String username, String password)` – це метод, який відповідає за отримання даних про користувача по його логіну та пароллю з Firestore;

- `public void createUser(User user)` – це метод, який відповідає за збереження даних про користувача в Firestore.

Клас CarRepository – це клас, який теж відповідає за отриманням даних від Firestore. Також відповідає за обробку даних та відправкою даних до CarViewModel. Основні його методи:

- `public LiveData<List<Car>> getCarByModel(String model)` – це метод, який відповідає за отримання списку всіх автомобілів по назві моделі або бренду з Firestore;

- `public void createDriverLicence(DriverLicence driverLicence)` – це метод, який відповідає за збереження даних про водійське посвідчення в Firestore;

- `public LiveData<List<Rent>> getUserRents()` – це метод, який відповідає за отримання списку оренд користувача з Firestore;

- `public void createRent(Rent rent)` – це метод, який відповідає за збереження даних про оренду в Firestore;

- `public void deleteRent(String idRent)` – це метод, який відповідає за видалення конкретної оренди з Firestore.

3.3.2 Пакет `viewmodel` та опис його класів

В пакеті `viewmodel` (рис. 3.5) містяться класи, які відповідають за бізнес-логіку застосунку.

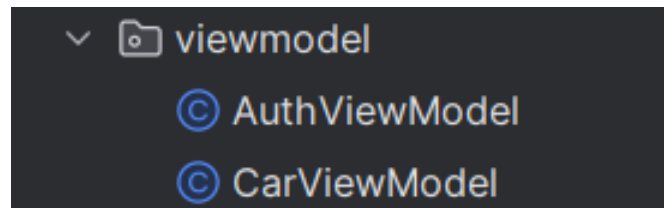


Рисунок 3.5 – Пакет `viewmodel`

Клас `AuthViewModel` є підкласом абстрактного класу `ViewModel`, який відповідає за бізнес-логіку застосунку, яка стосується авторизації, реєстрації та оновлення даних користувачів, отримання даних про користувачів. Основні його методи:

- `public LiveData<Boolean> preventFullName(String fullName)` – це метод, який відповідає за перевірку ПІБ на наявність цифр або спеціальних символів і повертає `true`, якщо немає ні цифр, ні спеціальних символів або повертає `false`, якщо є цифри або спеціальні символи;
- `public LiveData<Boolean> isEmailWriteCorrect(String email)` – це метод, який відповідає за перевірку електронної пошти на наявність `@gmail.com` і повертає `true`, якщо є `@gmail.com` або повертає `false`, якщо немає `@gmail.com`;
- `public LiveData<Boolean> isPasswordLargeThanOrEqualEight(String newPassword)` – це метод, який відповідає за перевірку пароля на кількість символів. Якщо кількість символів менше 8, то повертається `false` в іншому випадку повертається `true`;
- `public LiveData<Boolean> isPhoneWriteCorrect(String phone)` – це метод, який відповідає за перевірку номера телефона на наявність `+380` та кількість цифр після `+380`, яка повинна бути рівно 9. Повертає `false`, якщо

немає в наявності +380 або кількість цифр не дорівнює 9 в іншому випадку повертає true.

Клас `CarViewModel` є підкласом абстрактного класу `ViewModel`, який відповідає за бізнес-логіку застосунку, яка стосується автомобілів, оренд. Основні його методи:

- `public void createRent(Rent rent)` – це метод, який використовується для зберігання даних про оренду в базі даних, передаючи дані про оренду через метод використовуючи метод `createRent(rent)` класу `CarRepository`;
- `public LiveData<Boolean> isDriverLicenceNumberCorrect(String driverLicenceNumber)` – це метод, який відповідає за перевірку номера водійського посвідчення на кількість символів. Якщо кількість символів не дорівнює 9, то повертається false в іншому випадку повертається true;
- `public LiveData<Long> checkDate(String date)` – це метод, який відповідає за обчислення різниці між датою видачі та сьогоднішньою датою, передаючи до методу дату видачі автомобіля;
- `public LiveData<Integer> calculateDifferenceBetweenTwoDates(String startDate, String endDate, int price)` – це метод, який відповідає за обчислення різниці між датою повернення та видачі і на основі цієї різниці визначається загальна ціна оренди викликаючи метод `calculatePriceWithDifferenceInDays(long difference_In_Days, int price)`;
- `private Integer calculatePriceWithDifferenceInDays(long difference_In_Days, int price)` – це метод, який відповідає за обчислення загальної ціни оренди в залежності від значення різниці між датою повернення та видачі. Якщо значення цієї різниці знаходиться в діапазоні від 4 до 9, то ціна віднімається на 200, якщо значення цієї різниці знаходиться в діапазоні від 10 до 25, то ціна віднімається на 300, якщо значення цієї різниці більше або дорівнює 26, то ціна віднімається на 400.

3.3.3 Пакет view та опис його класів

В пакеті view (рис. 3.6) містяться класи, які відповідають за те, що пов'язано з UI: відображенням даних користувачеві, реагуванням на дії користувача, наприклад коли він натиснув на кнопку.

Інтерфейс RecyclerViewInterface описує класи, яким потрібно натискання на якийсь елемент та містить метод onItemClick(int position), який відповідає за це.

Класи CarAdapter, RentAdapter – це класи, які реалізують функціонал адаптера для відображення списку автомобілів, оренд у вигляді карток та натискання на конкретну картку.

Також вони мають клас ViewHolder, який успадковує клас RecyclerView.ViewHolder. Та реалізує метод onItemClick(position) інтерфейсу RecyclerViewInterface, що повертає позицію адаптера.

Клас RentFragment є підкласом класу Fragment, який відповідає за вікно «Мої оренди».

Клас HomeFragment є підкласом класу Fragment, який відповідає за головне меню застосунку. Його метод initView(List<Car> cars) відповідає за відображення списку автомобілів у формі карток використовуючи клас CarAdapter.

Клас AddDriverLicenceActivity є підкласом класу AppCompatActivity, який відповідає за вікно додавання водійського посвідчення клієнта.

Клас DetailActivity є підкласом класу AppCompatActivity, який відповідає за вікно детальної інформації про автомобіль.

Клас MainActivity є підкласом класу AppCompatActivity, який відповідає за вікно авторизації.

Клас OrderCarActivity є підкласом класу AppCompatActivity, який відповідає за вікно оформлення оренди.

Клас PaymentMethodActivity є підкласом класу AppCompatActivity, який відповідає за вікно оплати оренди.

Клас `RentDetailActivity` є підкласом класу `AppCompatActivity`, який відповідає за вікно детальної інформації про оренду.

Клас `SignUpActivity` є підкласом класу `AppCompatActivity`, який відповідає за вікно реєстрації.

Клас `SuccessfulActivity` є підкласом класу `AppCompatActivity`, який відповідає за вікно успішне завершення оплати.

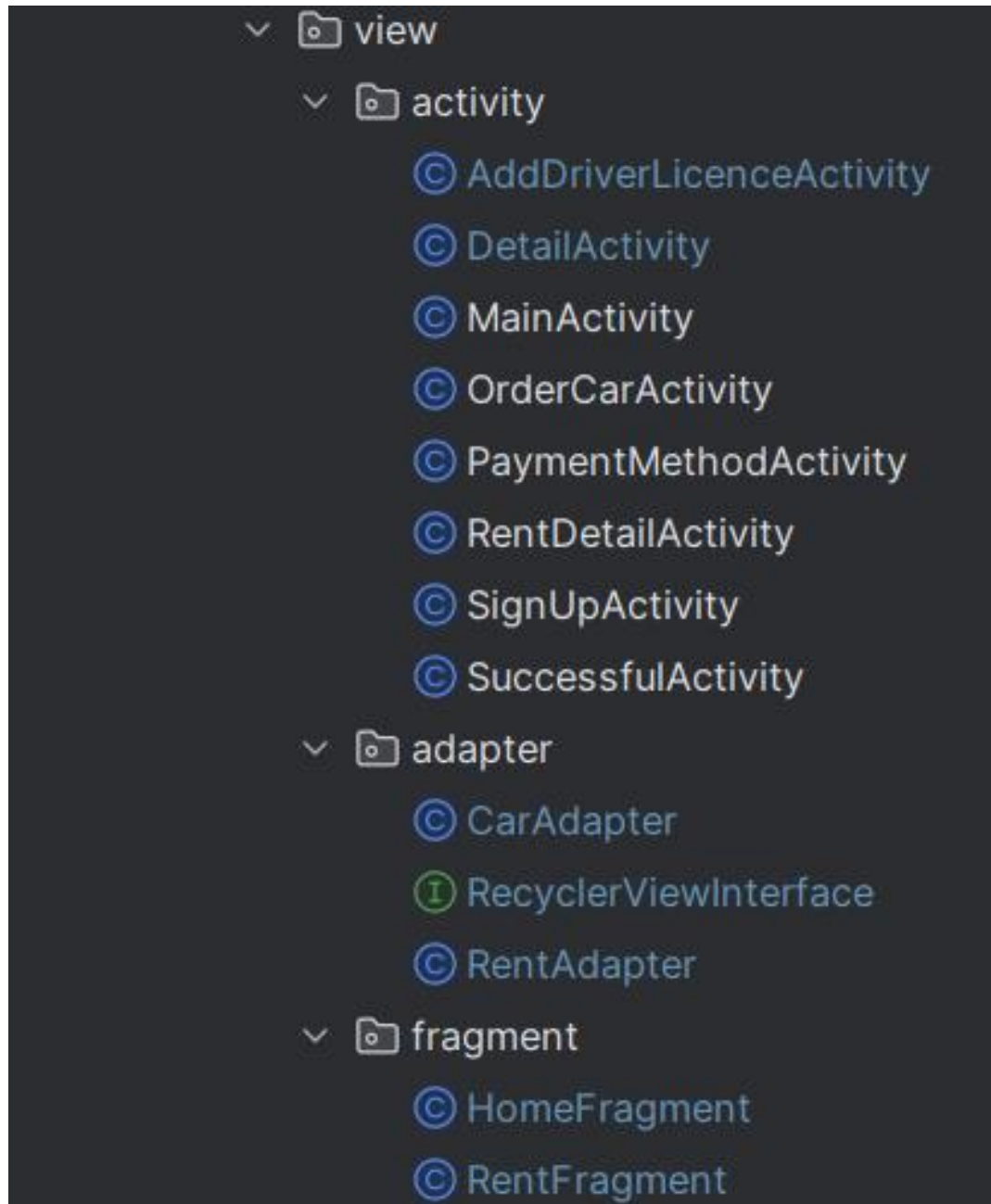


Рисунок 3.6 – Пакет view

3.4 Опис алгоритмів функціонування застосунку

3.4.1 Алгоритм реєстрації

Крок 1. Зчитуються введенні користувачем з текстових полів логін, ПІБ, електронна пошта, пароль та номер телефону.

Крок 2. Відправляється запит до Firestore на отримання даних про користувача на основі введеного логіну, передаючи логін через клас AuthViewModel до методу getUserByUsername класу AuthRepository.

Якщо користувач з таким логіном не існує, то перехід до кроку 3, в іншому випадку крок 5.

Крок 3. Перевіряється правильність введення ПІБ, електронної пошти, паролю та номеру телефона викликаючи методи preventFullName, isEmailWriteCorrect, isPasswordLargeThanOrEqualEight, isPhoneWriteCorrect класу AuthViewModel відповідно. Якщо вони повертають true, то перехід до кроку 4, в іншому випадку крок 5.

Крок 4. Відправляється запит до Firestore на зберігання даних про користувача, передаючи дані про користувача через клас AuthViewModel до методу createUser класу CarRepository.

Крок 5. Відображається користувачеві відповідне повідомлення про помилку.

Крок 6. Відбувається перехід користувача до вікна авторизації.

3.4.2 Алгоритм авторизації

Крок 1. Зчитується введенні користувачем з текстових полів логін та пароль.

Крок 2. Відправляється запит до Firestore на отримання даних про користувача на основі введеного логіну та паролю, передаючи логін та пароль через клас AuthViewModel до методу getUserByUsernameAndPassword класу

AuthRepository. Якщо такий користувач існує, то перехід до кроку 3, в іншому випадку перехід до кроку 4.

Крок 3. Відбувається перехід користувача до головного меню.

Крок 4. Відображається користувачеві відповідне повідомлення про помилку.

3.4.3 Алгоритм оформлення оренди

Крок 1. Зчитується введенні користувачем з текстових полів станція видачі та повернення, дата видачі та повернення.

Крок 2. Перевіряється текст дати видачі та кінцевий термін придатності на відмінність від тексту «Choose date». Якщо повертає true, то перехід до кроку 3, в іншому випадку крок 5.

Крок 3. Розраховується загальна сума оренди на основі різниці між датою повернення та видачі викликаючи метод `calculateDifferenceBetweenTwoDates` класу `CarViewModel`.

Крок 4. Відправляється запит до Firestore на зберігання даних про оренду, передаючи дані про оренду через клас `CarViewModel` до методу `createRent` класу `CarRepository`.

Крок 5. Відображається користувачеві відповідне повідомлення про помилку.

Крок 6. Відбувається перехід користувача до вікна оплати оренди.

3.4.4 Алгоритм додавання даних про водійське посвідчення

Крок 1. Зчитуються введенні користувачем з текстових полів номер водійського посвідчення, дати видачі та кінцевий термін придатності.

Крок 2. Перевіряється правильність введення номера водійського посвідчення викликаючи метод `isDriverLicenceNumberCorrect` класу `AuthViewModel`, також перевіряється текст дати видачі та кінцевий термін

придатності на відмінність від тексту «Choose date». Якщо вони повертають true, то перехід до кроку 3, в іншому випадку крок 4.

Крок 3. Відправляється запит до Firestore на зберігання даних про водійське посвідчення, передаючи дані про водійське посвідчення через клас CarViewModel до методу createDriverLicence класу CarRepository.

Крок 4. Відображається користувачеві відповідне повідомлення про помилку.

Крок 5. Відбувається перехід користувача до вікна оформлення оренди.

3.4.5 Алгоритм оплати оренди

Крок 1. Зчитуються введенні користувачем з текстових полів номер кредитної картки, ім'я власника, CVV та термін дії картки.

Крок 2. Користувач натискає на кнопку «Pay Now».

Крок 3. Зчитується введений користувачем з текстового поля спеціальний код для відправки грошей до загального рахунку.

Крок 4. Відправляється запит на підтвердження платежу.

Крок 5. Відбувається перехід користувача до вікна успішного завершення оплати.

3.4.6 Алгоритм скасування оренди

Крок 1. Перевіряється кількість днів між датою видачі автомобіля та сьогоднішньою викликаючи метод checkDate класу CarViewModel. Якщо різниця менше або дорівнює 2, то перехід до кроку 2.

Крок 2. Відправляється запит до Firestore на видалення оренди з бази даних, передаючи номер оренди через клас CarViewModel до методу deleteRent класу CarRepository.

Крок 3. Відображається користувачеві повідомлення про успішне скасування оренди.

Крок 4. Відбувається перехід користувача до головного меню.

3.4.7 Алгоритм пошуку автомобіля по бренду або моделі

Крок 1. Зчитується введений користувачем з текстового поля назва бренду або моделі автомобіля.

Крок 2. Відправляється запит до Firestore на отримання списку автомобілів, передаючи назву моделі або бренду через клас CarViewModel до методу getCarByModel класу CarRepository. Якщо є автомобілі з таким брендом або моделлю, то перехід до кроку 3, в іншому випадку крок 4.

Крок 3. Відображається список автомобілів в головному меню викликаючи метод initRecyclerView класу HomeFragment.

Крок 4. Відображається користувачеві повідомлення про відсутність автомобіля з такою назвою бренду або моделі.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення застосунку та умови його виконання

Застосунок призначений для того, щоб користувачі могли з легкістю брати автомобіль в оренду на будь-який термін часу, переглядати свої оренди, переглядати різну інформацію про автомобіль, шукати автомобіль за різними параметрами та інше.

Для забезпечення нормальної роботи застосунку необхідні такі мінімальні апаратні забезпечення:

- мобільний телефон з операційною системою Android версією не нижче 8.0;
- об'єм вільної пам'яті на телефоні не менше 120 Мб;
- доступ до мережі Інтернет.

4.2 Виконання роботи з застосунком

Для того, щоб запустити застосунок необхідно натиснути на іконку з назвою «Rent Car App». Якщо користувач вже був авторизований в системі, то відображається головне меню застосунку (рис. 4.1). В іншому випадку відображається вікно авторизації (рис. 4.2).

Якщо у користувача немає облікового запису, то його потрібно зареєструватися.

Для того, щоб користувач міг зареєструватися в системі необхідно натиснути на текст «Sign Up», після чого відображається вікно реєстрації, де користувач заповнює необхідну інформацію: логін, ПІБ, електронна пошта, пароль, номер телефону та фотографія користувача.

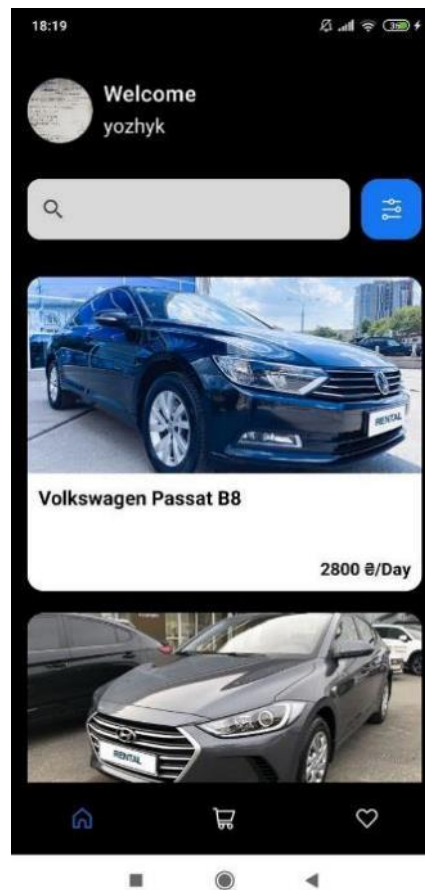


Рисунок 4.1 – Головне меню застосунку

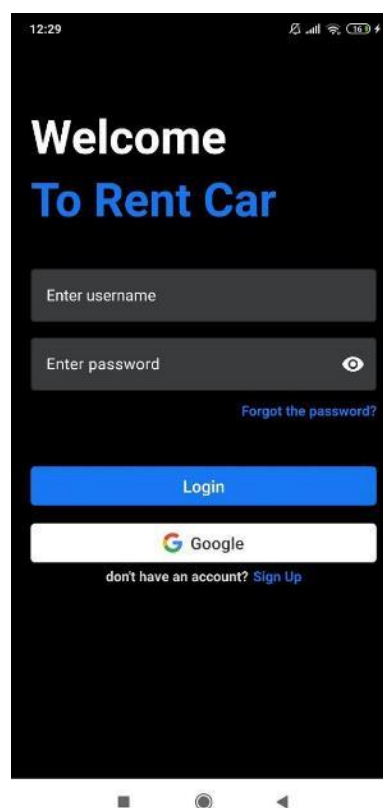


Рисунок 4.2 – Вікно авторизації

На рис. 4.3 наведено загальний вигляд вікна реєстрації.

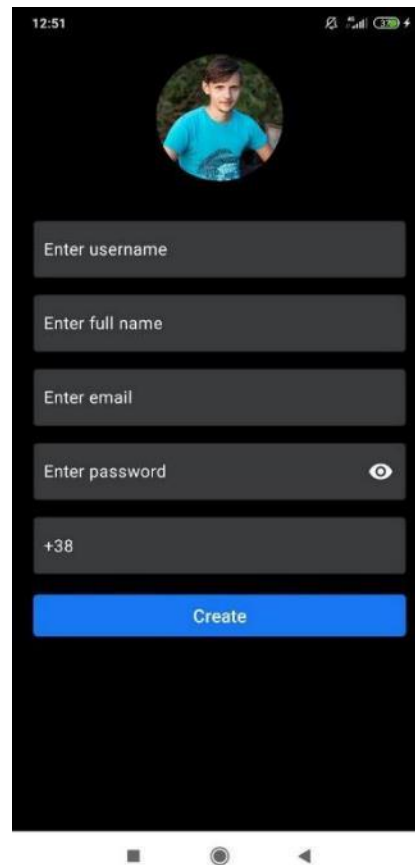


Рисунок 4.3 – Вікно реєстрації

Якщо користувач ввів логін, який вже зайнятий, або ввів ПІБ, електронну пошту, пароль або номер телефону неправильно, або не обрав фотографію, то з'являється відповідне повідомлення про помилку (рис. 4.4). Якщо користувач ввів все правильно та обрав фотографію, то він натискає на кнопку «Create», після чого відображається вікно авторизації.

Для того, щоб користувач міг авторизуватися в системі, йому потрібно ввести свій логін та пароль. Якщо користувач ввів логін, якого не існує в системі, або ввів пароль неправильно, то з'являється відповідне повідомлення про помилку (рис. 4.5). Після того, як користувач ввів всі необхідні дані для авторизації правильно, він натискає на кнопку «Login», після чого відображається головне меню застосунку.



12:30

The photo must be choose

Enter username
yozhyk

Such a username already exists

Enter full name
Yozhyk6

The full name must not contain any numbers or special characters

Enter email
yozhykovtolya

This mail does not have an @ or domain.

Enter password

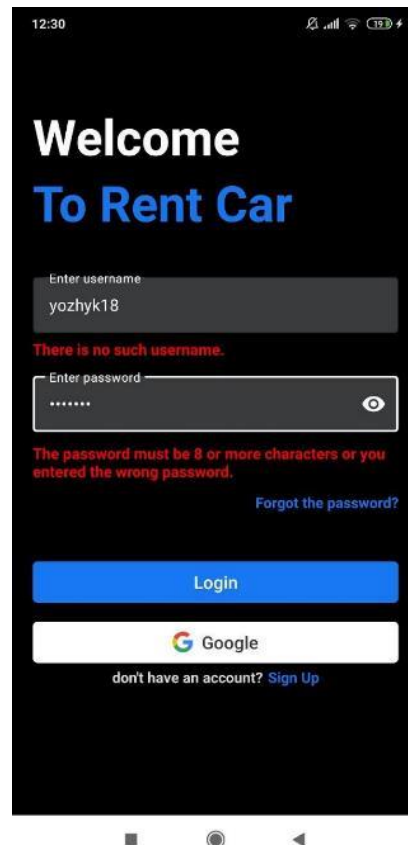
The password must be 8 or more characters

+38095304

The number of characters after +38 must be 9

Create

Рисунок 4.4 – Повідомлення про помилки під час реєстрації



12:30

Welcome To Rent Car

Enter username
yozhyk18

There is no such username.

Enter password

The password must be 8 or more characters or you entered the wrong password.

[Forgot the password?](#)

Login

 Google

don't have an account? [Sign Up](#)

Рисунок 4.5 – Повідомлення про помилки під час авторизації

Після того, як користувач потрапив до головного меню, він може знайти конкретний автомобіль, щоб переглянути детальну інформацію про нього або взяти в оренду.

Для того, щоб користувач міг знайти автомобіль через пошук, йому потрібно ввести частково або повністю назву моделі або бренду автомобіля, який він шукає. Наприклад, якщо ввести назву бренду «Toyota», то будуть відображені всі автомобілі, в яких бренд «Toyota» (рис. 4.6). Якщо автомобіль не було знайдено по назві моделі або бренду, відобразиться повідомлення «Sorry, but there is no such car.».

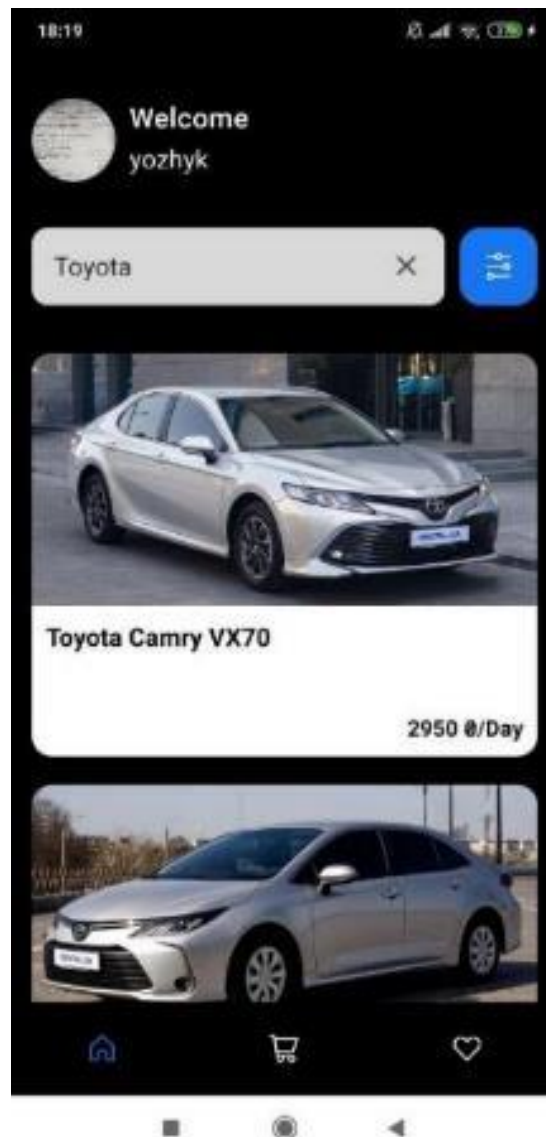


Рисунок 4.6 – Результати пошуку

Щоб обрати конкретний автомобіль, користувачу потрібно натиснути на автомобіль, який він бажає. Після цього з'явиться детальна інформація про цей автомобіль: його назва моделі, фотографії, характеристики та інше (рис. 4.7).

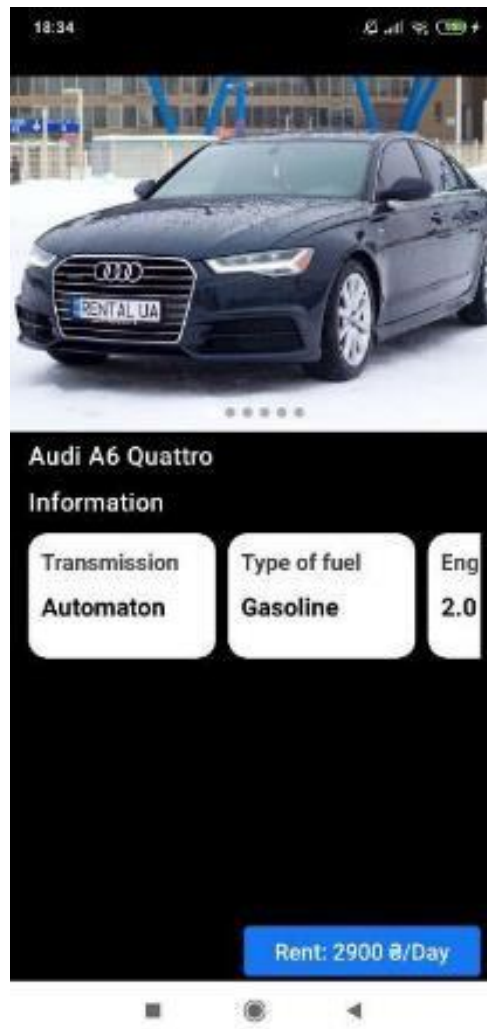


Рисунок 4.7 – Детальна інформація про автомобіль

Якщо користувач хоче взяти автомобіль в оренду, то для цього йому потрібно натиснути на кнопку «Rent», яка знаходиться в правому нижньому куті. Якщо він вперше бере в оренду автомобіль, то відобразиться вікно, де потрібно ввести інформацію про водійське посвідчення: номер посвідчення, дата видачі та чинний до (рис. 4.8). В іншому випадку відобразиться вікно оформлення оренди (рис. 4.9).

12:34

Get approved to drive

Since this is your first trip, you'll need to provide us information about your driver licence and credit card.

Driver Licence ID

xxxxxxxxxx

Date Issuing

Choose date

Until Date

Choose date

Next

Рисунок 4.8 – Вікно для заповнення інформації про водійське посвідчення

12:53

Select place, date and time

Station Issuing

Kharkiv, Kharkiv Airport

Station Return

Kharkiv, Kharkiv Airport

Date Issuing

Choose date

Date Return

Choose date

Payment Details	
Fines	0 ₾
Sum	0 ₾
Total Amount	0 ₾

Rent Car

Рисунок 4.9 – Вікно для оформлення оренди

Для того, щоб вибрати дату видачі та чинний до, потрібно натиснути на іконку календаря в полях з назвою «Choose date». Після цього відобразиться календар, де потрібно вибрати дату (рис. 4.10).



Рисунок 4.10 – Календар

Якщо користувач неправильно ввів номер посвідчення, або не обрав дату видачі та чинний до, то відображається відповідне повідомлення про помилку (рис. 4.11). Якщо користувач все зробив правильно, він натискає на кнопку «Next», після чого відображається вікно оформлення оренди.

13:34

Get approved to drive

Since this is your first trip, you'll need to provide us information about your driver licence and credit card.

Driver Licence ID

XXXXXXXXXX

Driver Licence ID must have only 9 symbols.

Date Issuing

Choose date

Choose the date issuing.

Until Date

Choose date

Choose the until date.

Next

Рисунок 4.11 – Повідомлення про помилки під час додавання інформації про водійське посвідчення

Для того, щоб оформити оренду, потрібно вибрати станцію, дату видачі та повернення. Після того, як користувач обере дату повернення автомобіля, буде підраховуватися вартість оренди в залежності від кількості днів оренди (рис. 4.12).

The screenshot shows a mobile application interface for car rental. At the top, the status bar displays the time 13:28, signal strength, and battery level. The main heading is "Select place, date and time". Below this, there are four selection fields: "Station Issuing" and "Station Return" both set to "Kharkiv, Kharkiv Airport", "Date Issuing" set to "5.4.2024", and "Date Return" set to "11.4.2024". Each field has a dropdown arrow or calendar icon. Below these fields is a "Payment Details" section. It contains a table with two rows: "Fines" with a value of "0 ₴" and "Sum" with a value of "2400 ₴". Below the table is a highlighted orange bar with "Total Amount" and "2400 ₴". At the bottom is a large blue button labeled "Rent Car". The Android navigation bar is visible at the very bottom.

13:28

Select place, date and time

Station Issuing

Kharkiv, Kharkiv Airport

Station Return

Kharkiv, Kharkiv Airport

Date Issuing

5.4.2024

Date Return

11.4.2024

Payment Details

Fines	0 ₴
Sum	2400 ₴

Total Amount 2400 ₴

Rent Car

Рисунок 4.12 – Вартість оренди

У випадку, якщо користувач не обрав дату видачі або повернення, відображається відповідне повідомлення про помилку (рис. 4.13).

12:53

Select place, date and time

Station Issuing

Kharkiv, Kharkiv Airport

Station Return

Kharkiv, Kharkiv Airport

Date Issuing

Choose date

Choose the date issuing rent.

Date Return

Choose date

Choose the date return rent

Payment Details

Fines	0 ₴
Sum	0 ₴
Total Amount	0 ₴

Rent Car

Рисунок 4.13 – Повідомлення про помилки під час оформлення оренди

Якщо користувач все оформив коректно, то він натискає на кнопку «Rent Car», де відобразиться вікно, в якому потрібно ввести номер картки, ім'я власника, CVV та термін дії картки (рис. 4.14). Якщо користувач ввів все коректно, то він натискає на кнопку «Pay Now», де відобразиться вікно, в якому потрібно підтвердити оплату ввівши спеціальний код, який він отримає (рис. 4.15).

12:55

← Rent Car ×

Add New Card

Card Number Expiry

Card Holder's name CVV

\$ 70 View Details Pay Now

Рисунок 4.14 – Кредитна картка

12:56

Razorpay Verified by VISA

PAYING TO Anatolii

TOTAL AMOUNT \$ 70

ENTER OTP

OTP sent to the phone number linked to your card ending with 1111.

OTP Submit →

Cancel Resend OTP

OR

Complete on bank's page →

This page will timeout after 02:56 minutes

Secured by Razorpay

Рисунок 4.15 – Підтвердження оплати

Після того, як користувач ввів спеціальний код та натиснув на кнопку «Submit», відображається вікно успішного завершення оплати (рис. 4.16). Після цього користувач натискає на кнопку «Back to Home», де він повертається до головного меню.

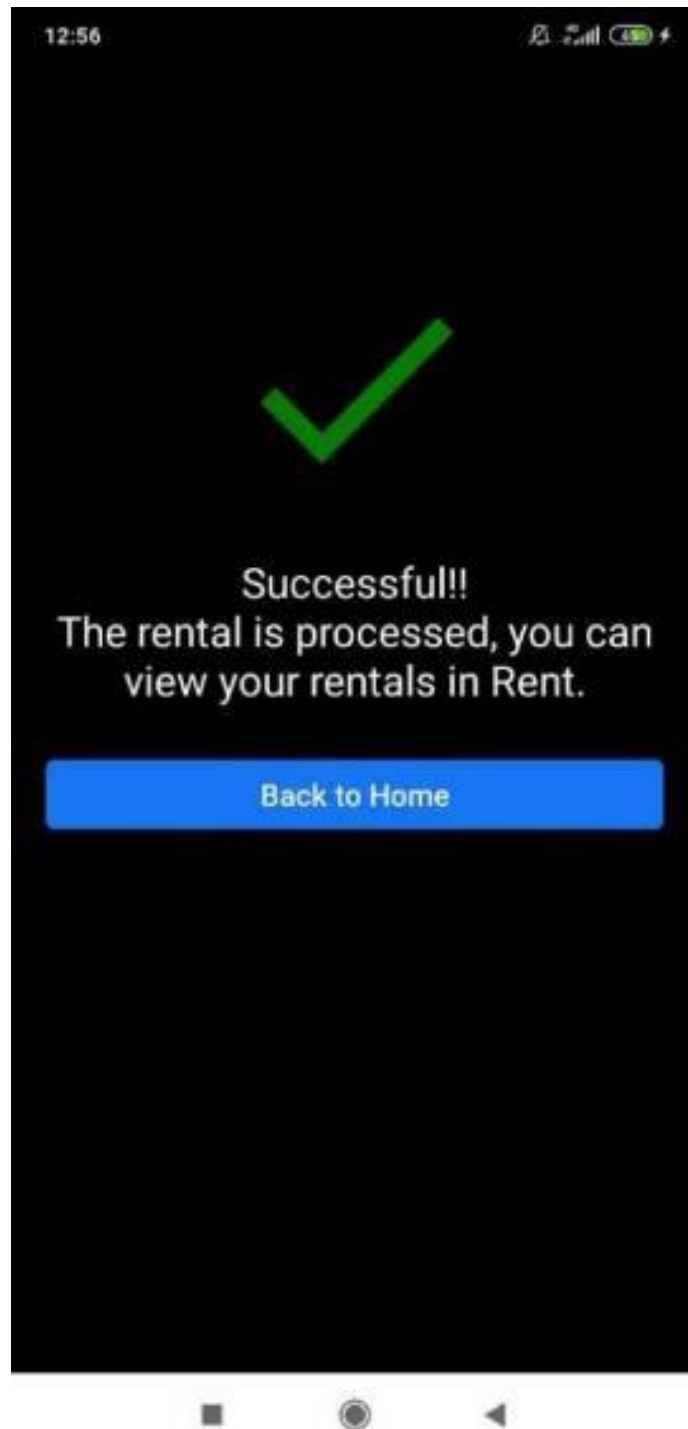


Рисунок 4.16 – Успішне завершення оплати

Якщо користувач бажає переглянути свої оренди, то йому потрібно натиснути в навігаційному меню на іконку зі кошиком, де відобразиться список його оренд (рис. 4.17).

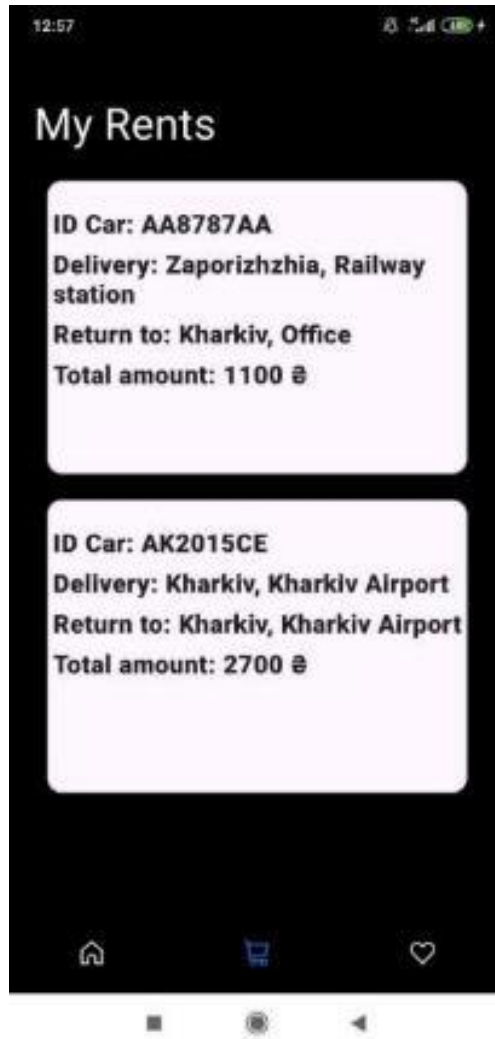


Рисунок 4.17 – Оренди користувача

Для того, щоб переглянути детальну інформацію про оренду, користувачу потрібно натиснути на конкретну оренду, після чого відобразиться вікно з детальною інформацією про неї (рис. 4.18).

Також користувач може скасувати оренду автомобіля, якщо кількість днів між датою видачі та сьогоднішньою не менше двох. У такому випадку він натискає на кнопку «Cancel Rent». В іншому випадку можливість скасувати оренду буде недоступною.

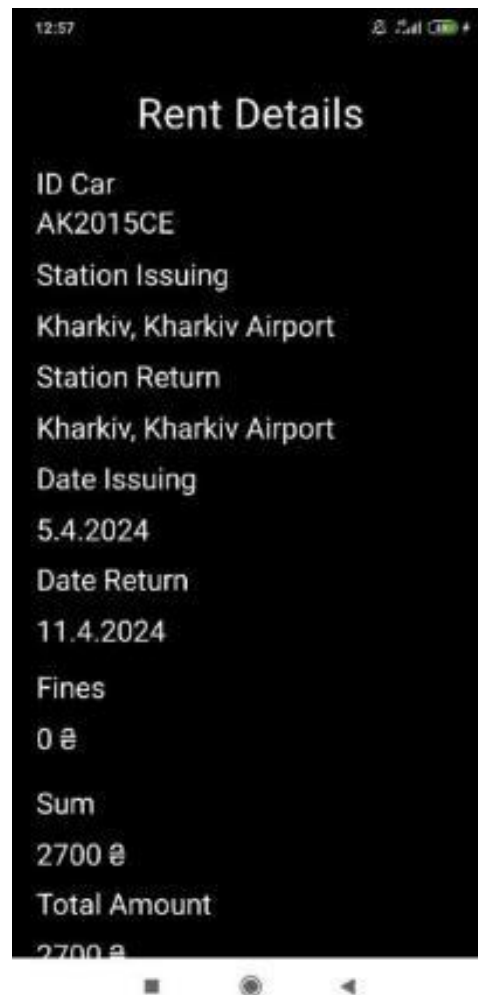


Рисунок 4.18 – Детальна інформація про оренду

4.3 Тестування застосунку та його результати

Для того, щоб перевірити якість застосунку було проведено ряд тестувань таких як:

- тестування верстки;
- функціональне тестування.

Для кожного тестування були створені чеклісти, де були зазначені на яких пристроях було проведено тестування, список перевірок та результати перевірки. На рис. 4.19 та 4.20 наведено чеклісти тестування верстки та функціонального тестування. Як результат, тестування було пройдено успішно і жодна із перевірок не отримала статус «Не пройдено».

Перевірка	Android (Xiaomi Mi 8 Lite)	Android (Samsung Galaxy A24)
Перевірити коректне відображення сторінок згідно файлів дизайну	Пройдено	Пройдено
Перевірити наявність головного меню	Пройдено	Пройдено
Перевірити наявність рядка пошуку	Пройдено	Пройдено
Перевірити стиль та коректне відображення шрифтів тексту та сам текст	Пройдено	Пройдено
Перевірити коректне відображення кнопок, блоків меню та ін.	Пройдено	Пройдено
Перевірити коректне відображення валюти	Пройдено	Пройдено
Перевірити орфографію/граматику контенту застосунку	Пройдено	Пройдено
Перевірити відповідність характеристик автомобіля	Пройдено	Пройдено
Перевірити коректне відображення пошуку	Пройдено	Пройдено
Перевірити коректне відображення цінового діапазону після зміни положення повзунка	Пройдено	Пройдено

Рисунок 4.19 – Чекліст тестування верстки

Перевірка	Android (Xiaomi Mi 8 Lite)	Android (Samsung Galaxy A24)
Перевірити реєстрацію в систему	Пройдено	Пройдено
Перевірити редагування профілю	Пройдено	Пройдено
Перевірити зміну паролю	Пройдено	Пройдено
Перевірити авторизацію в систему	Пройдено	Пройдено
Перевірити роботу пошуку за назвою автомобіля	Пройдено	Пройдено
Перевірити фільтрування за категоріями	Пройдено	Пройдено
Перевірити валідацію полів	Пройдено	Пройдено
Правильне вирахування вартості прокату після вибору дати видачі та повернення	Пройдено	Пройдено
Додати автомобіль до улюблених	Пройдено	Пройдено
Видалення автомобіля з улюблених	Пройдено	Пройдено
Перевірити авторизацію в систему через Google аккаунт	Пройдено	Пройдено
Перевірити додавання водійського посвідчення	Пройдено	Пройдено
Перевірити додавання банківської картки	Пройдено	Пройдено
Перевірити відображення валідаційних повідомлень:		
Невалідний логін	Пройдено	Пройдено
Невалідний пароль	Пройдено	Пройдено
Некоректна пошта	Пройдено	Пройдено
Пароль, який містить менше 8 символів	Пройдено	Пройдено
Некоректний номер телефона	Пройдено	Пройдено

Рисунок 4.20 – Чекліст функціонального тестування

ВИСНОВКИ

Під час виконання дипломної кваліфікаційної роботи було розроблено мобільний застосунок для взяття в оренду автомобіль.

Було виконано аналіз предметної області, існуючих аналогів, їхні переваги та недоліки, та визначено функціональні вимоги для застосунку.

Для розробки застосунку було обрано мову програмування Java, шаблон проєктування MVVM, сторонні сервіси Firestore та Razorpay.

Було спроектовано схему бази даних та імплементовано відповідні класи та методи.

Було проведено тестування застосунку, а саме: тестування верстки та функціональне тестування. І як результат, застосунок пройшов тестування успішно.

Створений застосунок має значну практичну цінність, оскільки спрощує процес оформлення та оплати оренди автомобіля, забезпечуючи широким вибором автомобіля, роблячи його більш зручним та доступним для користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Оренда автомобіля: особливості та основні переваги [Електрон. ресурс]. – Режим доступу: <https://hromadske.radio/publications/orenda-avtomobilia-osoblyvosti-ta-osnovni-perevahy>.
2. Прокат автомобілів [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BA%D0%B0%D1%82_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%BE%D0%B1%D1%96%D0%BB%D1%96%D0%B2.
3. Особливості та переваги автомобіля на прокат [Електрон. ресурс]. – Режим доступу: <https://ukr-prokat.com/blog/osoblyvosti-ta-perevagy-avtomobilya-na-prokat.html>.
4. Turo [Electronic resource]. – Access mode: https://play.google.com/store/apps/details?id=com.relayrides.android.relayrides&hl=en_US.
5. Find your drive at Turo, the world's largest car-sharing marketplace [Electronic resource]. – Access mode: <https://www.executivetraveller.com/turo-car-hire-rental-sharing-australia>.
6. Carngo Прокат авто [Електрон. ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=carngo.app&hl=uk&gl=US>.
7. Getaround [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Getaround>.
8. Getaround builds with Mapbox GL JS, Geocoding, and the Directions API [Electronic resource]. – Access mode: <https://www.mapbox.com/showcase/getaround>.
9. Java [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Java>.
10. Про мови програмування для Android [Електрон. ресурс]. – Режим доступу: <https://foxminded.ua/movy-prohramuvannia-dlia-android/>.

11. Jon Kowieski. Relational vs non-relational databases: Which one should you choose? [Electronic resource]. – Access mode: <https://www.thoughtspot.com/data-trends/data-modeling/relational-vs-non-relational-databases>.

12. Joseph Harisson. Difference Between Relational and Non Relational Database [Electronic resource]. – Access mode: <https://itcompanies.net/blog/relational-vs-non-relational-database>.

ДОДАТОК А
Технічне завдання

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується за темою «Розробка мобільного застосунку для оренди автомобілів» згідно з наказом №168 від 24 квітня 2024 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Застосунок призначений для того, щоб користувачі могли з легкістю брати автомобіль в оренду на будь-який термін часу, переглядати свої оренди, переглядати різну інформацію про автомобіль, шукати автомобіль за різними параметрами та інше.

A.3 Вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Застосунок повинен виконувати наступні функціональні вимоги:

- пошук автомобіля по назві моделі та бренду;
- перегляд детальної інформації про автомобіль;
- додавання автомобіля до улюблених;
- оформлення оренди;
- оплата оренди;
- перегляд власних оренд;
- перегляд детальної інформації про оренду;
- скасування оренди протягом двох днів;
- додавання даних про водійське посвідчення.

A.3.2 Вимоги до надійності

Застосунок повинен розпізнавати ситуації, коли вводяться некоректні дані для реєстрації, авторизації, додавання водійського посвідчення та виводити відповідне повідомлення про помилку. Також застосунок повинен розпізнавати ситуації, коли користувач вводить в пошук назву моделі або бренду автомобіля, якого немає та виводити повідомлення про відсутність такого автомобіля. Також при оформленні оренди, коли користувач не обрав дату видачі та повернення, та натиснув на кнопку повинно з'явитися повідомлення про помилку.

A.3.3 Вимоги до технічних та програмних засобів

Для забезпечення нормальної роботи застосунку необхідні такі мінімальні апаратні забезпечення:

- операційна система Android версією не нижче 8.0;
- об'єм вільної пам'яті на телефоні не менше 120 Мб;
- доступ до мережі Інтернет.

A.4 Порядок контролю та приймання

Дипломна робота допускається до захисту та захищається у присутності екзаменаційної комісії тільки після того, як вона була перевірена керівником, нормоконтролером, рецензентом та затверджена завідувачем кафедри.

ДОДАТОК Б
Текст програми

Б.1 Текст файла Car.java

```
package com.example.myapplication.rentcarapp.model.firestore.models;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class Car implements Serializable {
    private String ID;
    private String model;
    private String transmission;
    private String typeOfFuel;
    private int price;
    private List<String> photo = new ArrayList<>();
    private float avgRating;
    private String childrenChair;
    private double fuel;
    private String engineVolume;

    public Car(){}

    public String getID() {
        return ID;
    }

    public void setID(String ID) {
        this.ID = ID;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public String getTransmission() {
        return transmission;
    }
}
```

```

public void setTransmission(String transmission) {
    this.transmission = transmission;
}

public String getTypeOfFuel() {
    return typeOfFuel;
}

public void setTypeOfFuel(String typeOfFuel) {
    this.typeOfFuel = typeOfFuel;
}

public int getPrice() {
    return price;
}

public void setPrice(int price) {
    this.price = price;
}

public List<String> getPhoto() {
    return photo;
}

public void setPhoto(List<String> photo) {
    this.photo = photo;
}

public float getAvgRating() {
    return avgRating;
}

public void setAvgRating(float avgRating) {
    this.avgRating = avgRating;
}

public String getChildrenChair() {return childrenChair;}

public void setChildrenChair(String childrenChair)
{this.childrenChair = childrenChair;}

public double getFuel() {
    return fuel;
}

```

```

    }

    public void setFuel(double fuel) {
        this.fuel = fuel;
    }

    public String getEngineVolume() {
        return engineVolume;
    }

    public void setEngineVolume(String engineVolume) {
        this.engineVolume = engineVolume;
    }
}

```

Б.2 Текст файла DriverLicence.java

```

package com.example.myapplication.rentcarapp.model.firestore.models;

public class DriverLicence {
    private String id;
    private String dateIssuing;
    private String validDate;

    public DriverLicence() {}

    public DriverLicence(String id, String dateIssuing, String
validDate) {
        this.id = id;
        this.dateIssuing = dateIssuing;
        this.validDate = validDate;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
}

```

```

public String getDateIssuing() {
    return dateIssuing;
}

public void setDateIssuing(String dateIssuing) {
    this.dateIssuing = dateIssuing;
}

public String getValidDate() {
    return validDate;
}

public void setValidDate(String validDate) {
    this.validDate = validDate;
}
}

```

Б.3 Текст файла Rent.java

```

package com.example.myapplication.rentcarapp.model.firestore.models;

import java.io.Serializable;

public class Rent implements Serializable {
    private String id;
    private String client;
    private String car;
    private String stationIssuing;
    private String stationReturn;
    private String rentalDate;
    private String issuingDate;
    private String returnDate;
    private int fines;
    private int sum;
    private int totalAmount;
    private String status;

    public Rent() {}
}

```

```
public Rent(String id, String client, String car, String
stationIssuing, String stationReturn, String rentalDate, String issuingDate,
String returnDate, int sum, int totalAmount) {
    this.id = id;
    this.client = client;
    this.car = car;
    this.stationIssuing = stationIssuing;
    this.stationReturn = stationReturn;
    this.rentalDate = rentalDate;
    this.issuingDate = issuingDate;
    this.returnDate = returnDate;
    this.sum = sum;
    this.totalAmount = totalAmount;
}

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public String getClient() {
    return client;
}

public void setClient(String client) {
    this.client = client;
}

public String getCar() {
    return car;
}

public void setCar(String car) {
    this.car = car;
}

public String getStationIssuing() {
    return stationIssuing;
}
```

```
public void setStationIssuing(String stationIssuing) {
    this.stationIssuing = stationIssuing;
}

public String getStationReturn() {
    return stationReturn;
}

public void setStationReturn(String stationReturn) {
    this.stationReturn = stationReturn;
}

public String getRentalDate() {
    return rentalDate;
}

public void setRentalDate(String rentalDate) {
    this.rentalDate = rentalDate;
}

public String getIssuingDate() {
    return issuingDate;
}

public void setIssuingDate(String issuingDate) {
    this.issuingDate = issuingDate;
}

public String getReturnDate() {
    return returnDate;
}

public void setReturnDate(String returnDate) {
    this.returnDate = returnDate;
}

public int getFines() {
    return fines;
}

public void setFines(int fines) {
    this.fines = fines;
}
```

```

    }

    public int getSum() {
        return sum;
    }

    public void setSum(int sum) {
        this.sum = sum;
    }

    public int getTotalAmount() {
        return totalAmount;
    }

    public void setTotalAmount(int totalAmount) {
        this.totalAmount = totalAmount;
    }

    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }
}

```

Б.4 Текст файла Station.java

```

package com.example.myapplication.rentcarapp.model.firestore.models;

public class Station {
    private String name;
    private String address;

    public Station() {}

    public String getName() {
        return name;
    }
}

```

```

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}

```

Б.5 Текст файла User.java

```

package com.example.myapplication.rentcarapp.model.firestore.models;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class User implements Serializable {
    private String token;
    private String fullName;
    private String phoneNumber;
    private String email;
    private String driverLicence;
    private String photo;
    private List<String> cards = new ArrayList<>();
    private List<String> cars = new ArrayList<>();

    private String username;
    private String password;

    public User() {}

    public User(String token, String fullName, String phoneNumber,
String email, String driverLicence, String photo, List<String> cards,
List<String> cars, String username, String password) {
        this.token = token;
        this.fullName = fullName;

```

```
        this.phoneNumber = phoneNumber;
        this.email = email;
        this.driverLicence = driverLicence;
        this.photo = photo;
        this.cards = cards;
        this.cars = cars;
        this.username = username;
        this.password = password;
    }

    public String getToken() {
        return token;
    }

    public void setToken(String token) {
        this.token = token;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getDriverLicence() {
```

```
        return driverLicence;
    }

    public void setDriverLicence(String driverLicence) {
        this.driverLicence = driverLicence;
    }

    public String getPhoto() {
        return photo;
    }

    public void setPhoto(String photo) {
        this.photo = photo;
    }

    public List<String> getCards() {
        return cards;
    }

    public void setCards(List<String> cards) {
        this.cards = cards;
    }

    public List<String> getCars() {
        return cars;
    }

    public void setCars(List<String> cars) {
        this.cars = cars;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }
}
```

```

        public void setPassword(String password) {
            this.password = password;
        }
    }
}

```

Б.6 Текст файла SuccessfulActivity.java

```

package com.example.myapplication.rentcarapp.view.activity;

import androidx.appcompat.app.AppCompatActivity;

import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.ConnectivityManager;
import android.os.Bundle;
import android.view.View;

import com.example.myapplication.rentcarapp.R;
import com.example.myapplication.rentcarapp.receiver.InternetReceiver;

import dagger.hilt.android.AndroidEntryPoint;

@AndroidEntryPoint
public class SuccessfulActivity extends AppCompatActivity {
    BroadcastReceiver broadcastReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_successful);
    }

    @Override
    protected void onResume() {
        super.onResume();
        initBroadcastReceiver();
    }

    private void initBroadcastReceiver(){
        broadcastReceiver = new InternetReceiver();
    }
}

```

```

        registerReceiver(broadcastReceiver,
IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));
    }

    public void backToHome(View view){
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(broadcastReceiver);
    }
}

```

Б.7 Текст файла SignUpActivity.java

```

package com.example.myapplication.rentcarapp.view.activity;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProvider;

import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.ConnectivityManager;
import android.net.Uri;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import com.example.myapplication.rentcarapp.R;
import
com.example.myapplication.rentcarapp.model.firestore.models.Client;

```

```

import
com.example.myapplication.rentcarapp.model.firestore.models.User;
import com.example.myapplication.rentcarapp.receiver.InternetReceiver;
import com.example.myapplication.rentcarapp.viewmodel.AuthViewModel;
import com.google.android.material.imageview.ShapeableImageView;
import com.google.android.material.textfield.TextInputEditText;

import org.checkerframework.checker.units.qual.A;

import java.util.Objects;
import java.util.concurrent.atomic.AtomicBoolean;

import dagger.hilt.android.AndroidEntryPoint;

@AndroidEntryPoint
public class SignUpActivity extends AppCompatActivity {
    ShapeableImageView icon;
    Uri uri;
    AuthViewModel authViewModel;
    TextInputEditText newUsername, newFullName, newEmail, userPassword,
newPhone;
    TextView incorrectFullName, incorrectEmail, incorrectPassword,
incorrectPhone, existUsername, choosePhoto;
    BroadcastReceiver broadcastReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sign_up);
        initComponents();
        authViewModel = new
ViewModelProvider(this).get(AuthViewModel.class);
        editUsername();
        editFullName();
        editEmail();
        editPassword();
        editPhoneNumber();
    }

    @Override
    protected void onResume() {
        super.onResume();
        initBroadcastReceiver();
    }

```

```

    }

    private void initComponents() {
        icon = findViewById(R.id.avatar);
        newUsername = findViewById(R.id.newUsername);
        newFullName = findViewById(R.id.newFullName);
        newEmail = findViewById(R.id.newEmail);
        userPassword = findViewById(R.id.userPassword);
        newPhone = findViewById(R.id.newPhone);
        initErrorMessages();
    }

    private void initErrorMessages() {
        existUsername = findViewById(R.id.existUsername);
        incorrectFullName = findViewById(R.id.incorrectFullName);
        incorrectEmail = findViewById(R.id.incorrectEmail);
        incorrectPassword = findViewById(R.id.incorrectPassword);
        incorrectPhone = findViewById(R.id.incorrectPhone);
        choosePhoto = findViewById(R.id.choosePhoto);
    }

    private void initBroadcastReceiver() {
        broadcastReceiver = new InternetReceiver();
        registerReceiver(broadcastReceiver,
            new
            IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));
    }

    public void pickUpImage(View view) {
        Intent intent = new Intent(Intent.ACTION_PICK);
        startActivityForResult(intent, 1000);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode,
        @Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if(resultCode == RESULT_OK) {
            if(requestCode == 1000) {
                if(data != null) {
                    uri = data.getData();
                    icon.setImageURI(data.getData());
                }
            }
        }
    }

```

```

        }
    }

    private void editUsername(){
        newUsername.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int
count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int
before, int count) {
                String username =
Objects.requireNonNull(newUsername.getText()).toString();
                checkUsername(username);
            }

            @Override
            public void afterTextChanged(Editable s) {}
        });
    }

    private void editFullName(){
        newFullName.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int
count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int
before, int count) {
                String fullName =
Objects.requireNonNull(newFullName.getText()).toString();
                if(!checkFullName(fullName)){
                    incorrectFullName.setVisibility(View.VISIBLE);
                }else{
                    incorrectFullName.setVisibility(View.GONE);
                }
            }

            @Override
            public void afterTextChanged(Editable s) {}
        });
    }

```

```

    }

    private void editEmail(){
        newEmail.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int
count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int
before, int count) {
                String email =
Objects.requireNonNull(newEmail.getText()).toString();
                if(!checkEmail(email)){
                    incorrectEmail.setVisibility(View.VISIBLE);
                }else{
                    incorrectEmail.setVisibility(View.GONE);
                }
            }

            @Override
            public void afterTextChanged(Editable s) {}
        });
    }

    private void editPassword(){
        userPassword.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int
count, int after) {}

            @Override
            public void onTextChanged(CharSequence s, int start, int
before, int count) {
                String password =
Objects.requireNonNull(userPassword.getText()).toString();
                if(!checkPassword(password)){
                    incorrectPassword.setVisibility(View.VISIBLE);
                }else{
                    incorrectPassword.setVisibility(View.GONE);
                }
            }
        })
    }

```

```

        @Override
        public void afterTextChanged(Editable s) {}
    });
}

private void editPhoneNumber() {
    newPhone.addTextChangedListener(new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence s, int start, int
count, int after) {}

        @Override
        public void onTextChanged(CharSequence s, int start, int
before, int count) {
            String phone =
Objects.requireNonNull(newPhone.getText()).toString();
            if(!checkPhoneNumber(phone)) {
                incorrectPhone.setVisibility(View.VISIBLE);
            }else{
                incorrectPhone.setVisibility(View.GONE);
            }
        }
    });

    @Override
    public void afterTextChanged(Editable s) {}
}

private void checkUsername(String username) {
    authViewModel.getUserByUsername(username).observe(this, user ->
{
        if(user != null){
            existUsername.setVisibility(View.VISIBLE);
        }else{
            existUsername.setVisibility(View.GONE);
        }
    });
}

private boolean checkFullName(String fullName) {
    AtomicBoolean isFullNameCorrect = new AtomicBoolean(false);
    authViewModel.isFullNameWriteCorrect(fullName).observe(this,
aBoolean -> {

```

```

        if(aBoolean){
            isFullNameCorrect.set(true);
        }
    });
    return isFullNameCorrect.get();
}

private boolean checkEmail(String email){
    AtomicBoolean isEmailWriteCorrect = new AtomicBoolean(false);
    authViewModel.isEmailCorrect(email).observe(this, aBoolean -> {
        if(aBoolean){
            isEmailWriteCorrect.set(true);
        }
    });
    return isEmailWriteCorrect.get();
}

private boolean checkPassword(String password){
    AtomicBoolean isPasswordCorrect = new AtomicBoolean(false);
    authViewModel.isPasswordWriteCorrect(password).observe(this,
aBoolean -> {
        if(aBoolean){
            isPasswordCorrect.set(true);
        }
    });
    return isPasswordCorrect.get();
}

private boolean checkPhoneNumber(String phone){
    AtomicBoolean isPhoneNumberWriteCorrect = new
AtomicBoolean(false);
    authViewModel.isUserPhoneCorrect(phone).observe(this, aBoolean
-> {
        if(aBoolean){
            isPhoneNumberWriteCorrect.set(true);
        }
    });
    return isPhoneNumberWriteCorrect.get();
}

private void showErrorMessages(){
    choosePhoto.setVisibility(View.VISIBLE);
    incorrectFullName.setVisibility(View.VISIBLE);
}

```

```

        incorrectEmail.setVisibility(View.VISIBLE);
        incorrectPassword.setVisibility(View.VISIBLE);
        incorrectPhone.setVisibility(View.VISIBLE);
    }

    private void hideErrorMessages() {
        choosePhoto.setVisibility(View.GONE);
        incorrectFullName.setVisibility(View.GONE);
        incorrectEmail.setVisibility(View.GONE);
        incorrectPassword.setVisibility(View.GONE);
        incorrectPhone.setVisibility(View.GONE);
    }

    public void createAccount(View view) {
        String username =
Objects.requireNonNull(newUsername.getText()).toString();
        authViewModel.getUserByUsername(username).observe(this, user ->
{
            if(user == null) {
                createUser(username);
                existUsername.setVisibility(View.GONE);
            }else{
                existUsername.setVisibility(View.VISIBLE);
            }
        });
    }

    private void createUser(String username) {
        String email =
Objects.requireNonNull(newEmail.getText()).toString();
        String password =
Objects.requireNonNull(userPassword.getText()).toString();
        String phone =
Objects.requireNonNull(newPhone.getText()).toString();
        String fullName =
Objects.requireNonNull(newFullName.getText()).toString();
        if (checkFullName(fullName) && checkPhoneNumber(phone) &&
checkEmail(email) && checkPassword(password) && uri != null) {
            hideErrorMessages();
            User newUser = new User(username, password);
            createClient(newUser, fullName, email, password, phone);
        }else{
            showErrorMessages();

```

```

        }
    }

    private void createClient(User newUser, String fullName, String
email, String password, String phone);
        authViewModel.signUpWithEmailAndPassword(newClient,      newUser,
password);

        goToMainActivity();
    }

    private void goToMainActivity(){
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(broadcastReceiver);
    }
}

```

Б.8 Текст файла RentDetailActivity.java

```

package com.example.myapplication.rentcarapp.view.activity;

import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProvider;

import android.content.BroadcastReceiver;
import android.content.Intent;
import android.content.IntentFilter;
import android.graphics.Color;
import android.net.ConnectivityManager;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import com.example.myapplication.rentcarapp.R;

```

```

import
com.example.myapplication.rentcarapp.model.firestore.models.Rent;
import com.example.myapplication.rentcarapp.receiver.InternetReceiver;
import com.example.myapplication.rentcarapp.viewmodel.CarViewModel;
import com.google.android.material.button.MaterialButton;
import com.google.android.material.textfield.TextInputEditText;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

import dagger.hilt.android.AndroidEntryPoint;

@AndroidEntryPoint
public class RentDetailActivity extends AppCompatActivity {
    Rent rent;
    MaterialButton returnCar, cancelRent;
    TextView rentIDCar, issuingStationRent, returnStationRent,
        dateIssuingRent, dateReturnRent, finesRent,
        sumRent, amountRent, statusRent;
    CarViewModel carViewModel;
    BroadcastReceiver broadcastReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rent_detail);
        rent = (Rent) getIntent().getSerializableExtra("Rent");
        carViewModel = new
ViewModelProvider(this).get(CarViewModel.class);
        initOther();
        initStation();
        initDate();
        initPrice();
        checkDateToReturnCar();
        checkDateToIssuingCar();
    }

    @Override
    protected void onResume() {
        super.onResume();
        initBroadcastReceiver();
    }

```

```

    }

    private void initBroadcastReceiver() {
        broadcastReceiver = new InternetReceiver();
        registerReceiver(broadcastReceiver,
            new
            IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION));
    }

    private void initStation() {
        issuingStationRent = findViewById(R.id.issuingStationRent);
        returnStationRent = findViewById(R.id.returnStationRent);
        issuingStationRent.setText(rent.getStationIssuing());
        returnStationRent.setText(rent.getStationReturn());
    }

    private void initDate() {
        dateIssuingRent = findViewById(R.id.dateIssuingRent);
        dateReturnRent = findViewById(R.id.dateReturnRent);
        dateIssuingRent.setText(rent.getIssuingDate());
        dateReturnRent.setText(rent.getReturnDate());
    }

    private void initPrice() {
        finesRent = findViewById(R.id.finesRent);
        sumRent = findViewById(R.id.sumRent);
        amountRent = findViewById(R.id.amountRent);
        finesRent.setText(rent.getFines() + " €");
        sumRent.setText(rent.getSum() + " €");
        amountRent.setText(rent.getTotalAmount() + " €");
    }

    private void initOther() {
        rentIDCar = findViewById(R.id.rentIDCar);
        statusRent = findViewById(R.id.statusRent);
        returnCar = findViewById(R.id.returnCar);
        cancelRent = findViewById(R.id.cancelRent);
        rentIDCar.setText(rent.getCar());
        statusRent.setText(rent.getStatus());
        returnCar.setEnabled(false);
    }

    private void checkDateToReturnCar() {

```

```

        carViewModel.checkDate(rent.getReturnDate()).observe(this,
difference_In_Days -> {
            isDateToReturnCar(difference_In_Days);
            checkRentOnFine(difference_In_Days);
        });
    }

    private void checkDateToIssuingCar() {
        carViewModel.checkDate(rent.getIssuingDate()).observe(this,
this::isCanCancelRent);
    }

    private void isCanCancelRent(long difference_In_Days) {
        if(difference_In_Days <= -2) {
            cancelRent.setEnabled(false);
            cancelRent.setVisibility(View.GONE);
        }
    }

    public void returnRent(View view) {
        if(rent.getFines() > 0) {
            paymentOfThePenalty(rent.getFines());
        } else {
            showMessageAboutRent("Rent " + rent.getId() + " was
deleted");

            returnToMainWindow();
        }
    }

    private void paymentOfThePenalty(int fines) {
        showMessageAboutRent("Rent " + rent.getId() + " was deleted");
        goToPaymentMethodActivity(fines);
    }

    public void onCancelRent(View view) {
        showMessageAboutRent("Rent " + rent.getId() + " was canceled");
        returnToMainWindow();
    }

    private void goToPaymentMethodActivity(int fines) {
        Intent intent = new Intent(this, PaymentMethodActivity.class);
        intent.putExtra("Amount", fines);
        startActivity(intent);
    }

```

```

    }

    private void showMessageAboutRent(String message){
        carViewModel.updateRentStatus(rent.getId(), "Returned");
        carViewModel.deleteRent(rent.getId());
        Toast.makeText(getApplicationContext(), message,
Toast.LENGTH_LONG).show();
    }

    private void returnToMainWindow(){
        Intent intent = new Intent(this, MainWindowActivity.class);
        startActivity(intent);
    }

    private void isDateToReturnCar(long difference_In_Days){
        Log.i("Time", String.valueOf(difference_In_Days));
        if(difference_In_Days <= 0){
            returnCar.setEnabled(true);
        }
    }

    private void checkRentOnFine(long difference_In_Days){
        if(difference_In_Days <= -1){
            int fine = rent.getFines() + 100;
            carViewModel.updateFineRent(rent.getId(), fine);
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        unregisterReceiver(broadcastReceiver);}}

```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Слайд 1

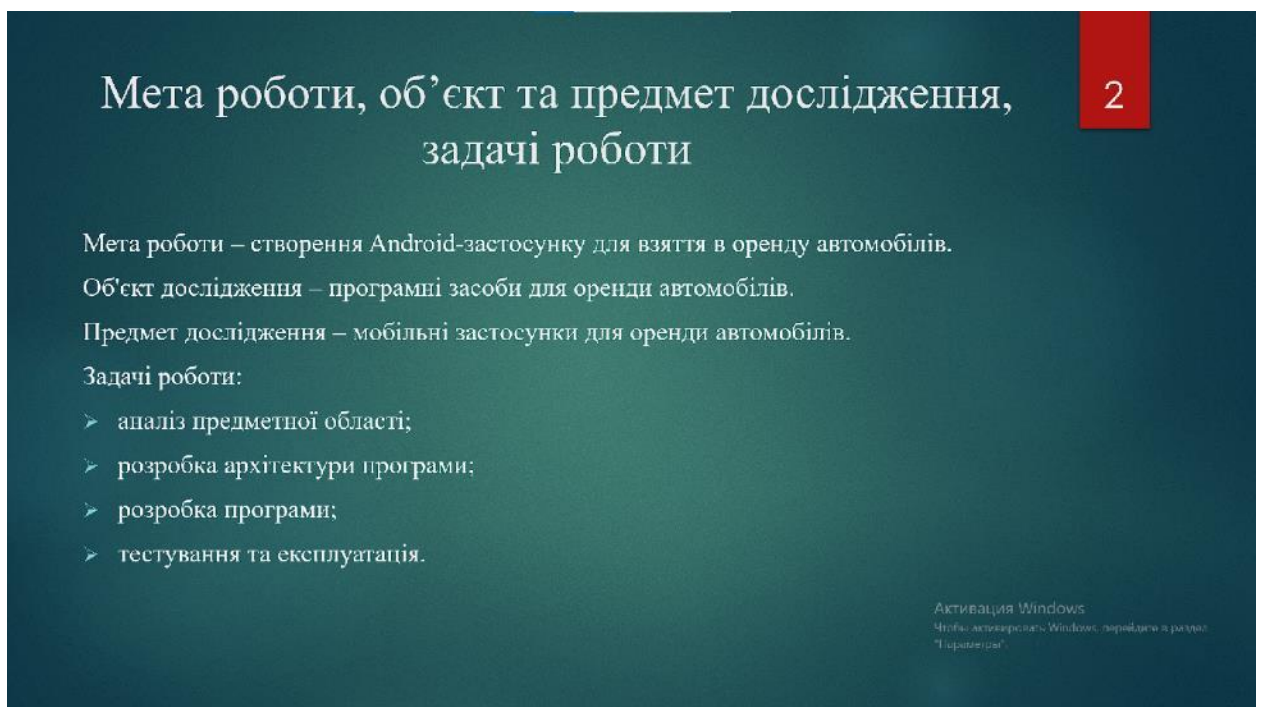


Рисунок В.2 – Слайд 2

Порівняння існуючих аналогів

3

Критерій порівняння	Turo	Carngo	Getaround
Доступність	Канада, Великобританія	Канада	Європа
Проблема зі зворотнім зв'язком	-	-	-
Широкий вибір автомобілю	+	-	+
Зручність	+	-	-
Гнучкість при виборі авто, локації та часу оренди	+	-	-

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.3 – Слайд 3

Вибір мови програмування

4

Критерій порівняння	Java	Kotlin	C++
Продуктивність	Висока	Висока	Середня
Складність розробки	Середня	Середня	Висока
Безпека та надійність	Висока	Середня	Низька
Підтримка спільноти	Висока	Висока	Середня

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.4 – Слайд 4

Вибір сервісу для побудови бази даних

5

Опис	Realtime Database	Firestore
Модель даних	Зберігає дані у вигляді дерева JSON	Зберігає дані в документах
Масштабованість	Має певний ліміт в масштабуванні	Автоматично масштабується
Запити	Підтримує глибокі запити зі обмеженими можливостями сортування та фільтрування	Підтримує розширені індексовані запити зі складним сортуванням і фільтруванням

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.5 – Слайд 5

Вибір сервісу для оплати оренди

6

Критерій порівняння	Razorpay	PayPal	Stripe
Безпека	Висока	Середня	Середня
Простота налаштування та використання	Висока	Висока	Середня
Широкий вибір способу оплати	+	+	-

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.6 – Слайд 6



Рисунок В.7 – Слайд 7



Рисунок В.8 – Слайд 8

Схема функціонування застосунку

9

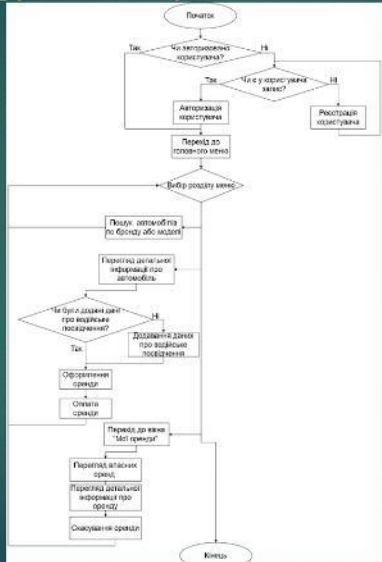


Рисунок 3 – Схема функціонування застосунку

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.9 – Слайд 9

Схема бази даних та зв'язки між сутностями

10

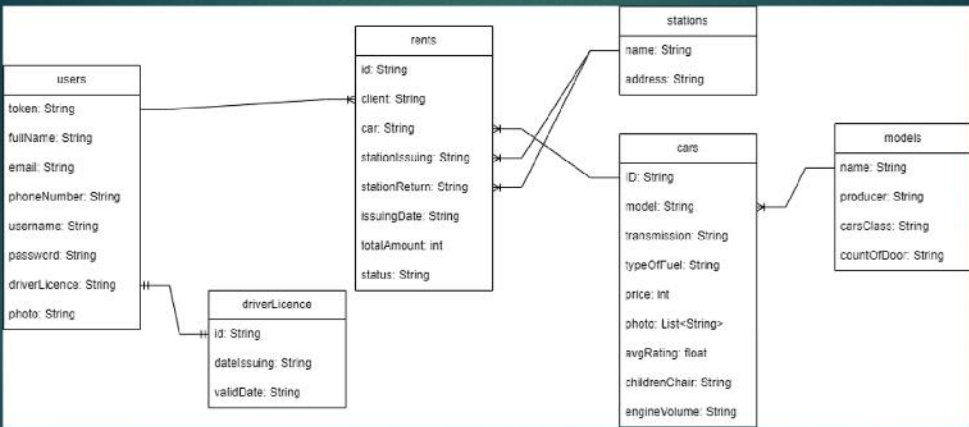


Рисунок 4 – Схема бази даних та зв'язки між сутностями

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.10 – Слайд 10

Розроблені класи застосунку

- model
 - firestore
 - models
 - Car
 - DriverLicence
 - Rent
 - Station
 - User
 - repository
 - AuthRepository
 - CarRepository

- viewmodel
 - AuthViewModel
 - CarViewModel

- view
 - activity
 - AddDriverLicenceActivity
 - DetailActivity
 - MainActivity
 - OrderCarActivity
 - PaymentMethodActivity
 - RentDetailActivity
 - SignUpActivity
 - SuccessfulActivity
 - adapter
 - CarAdapter
 - RecyclerViewInterface
 - RentAdapter
 - fragment
 - HomeFragment
 - RentFragment

Рис. 5 – Пакет model та його класи

Рис. 6 – Пакет model та його класи

Рис. 7 – Пакет view та його класи

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.11 – Слайд 11

Інтерфейси застосунку



Рисунок 8 – Вікно авторизації



Рисунок 9 – Вікно реєстрації



Рисунок 10 – Головне меню



Рисунок 11 – Пошук автомобіля



Рисунок 12 – Детальна інформація про автомобіль

Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

Рисунок В.12 – Слайд 12



Висновки

15

Під час виконання дипломної кваліфікаційної роботи було розроблено мобільний застосунок для взяття в оренду автомобіль.

Було виконано аналіз предметної області, існуючих аналогів, їхні переваги та недоліки, та визначено функціональні вимоги для застосунку.

Для розробки застосунку було обрано мову програмування Java, шаблон проєктування MVVM, сторонні сервіси Firestore та Razorpay.

Було спроектовано схему бази даних та імплементовано відповідні класи та методи.

Було проведено тестування застосунку, а саме: тестування верстки та функціональне тестування. І як результат, застосунок пройшов тестування успішно.

Створений застосунок має значну практичну цінність, оскільки спрощує процес оформлення та оплати оренди автомобіля, забезпечуючи широким вибором автомобіля, роблячи його більш зручним та доступним для користувача.

Активация Windows
Чтобы активировать Windows, перейдите в раздел
"Параметры".

Рисунок В.15 – Слайд 15