ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для пошуку та порівняння ціни автомобілів мовою Kotlin»

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

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

_____ Віталій ВДОВИЧЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Віталій ВДОВИЧЕНКО

Керівник: Володимир САДОВЕНКО
к.ф.-м.н., доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Вдовиченку Віталію Миколайовичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для пошуку та порівняння ціни автомобілів мовою Kotlin»
керівник кваліфікаційної роботи к.ф.-м.н., доцент Володимир САДОВЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про розробку мобільних застосунків, мова програмування Kotlin, набір інструментів Android Jetpack, набір інструментів Firebase.

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

1. Формування вимог до мобільного застосунку для пошуку та порівняння ціни автомобілів
2. Проектування мобільного застосунку для пошуку та порівняння ціни автомобілів.
3. Програмна реалізація та опис функціонування мобільного застосунку для пошуку та порівняння ціни автомобілів.

4. Тестування мобільного застосунку.

5. Перелік графічного матеріалу:
1. Аналіз аналогів.
2. Вимоги до застосунку.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Архітектура застосунку.
6. Діаграма класів.
7. Структура JSON файлу.
8. Екранні форми.
9. Демонстрація застосунку.
10. Апробація результатів дослідження.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Формування вимог до мобільного застосунку для пошуку та порівняння ціни автомобілів	14.03.-15.03.2024	
4	Проектування мобільного застосунку для пошуку та порівняння ціни автомобілів	16.03.-18.03.2024	
5	Програмна реалізація мобільного застосунку для пошуку та порівняння ціни автомобілів	19.03.-20.04.2024	
6	Тестування мобільного застосунку	21.04.-29.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	30.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Віталій ВДОВИЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Володимир САДОВЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 50 стор., 2 табл., 38 рис., 28 джерел.

Мета роботи – спрощення процесу пошуку та порівняння автомобілів за рахунок застосування мобільного додатку, створеного на мові Kotlin.

Об'єкт дослідження – процес пошуку та порівняння ціни автомобілів.

Предмет дослідження – програмне забезпечення для пошуку та порівняння ціни автомобілів.

Короткий зміст роботи: В роботі сформовано вимоги до мобільного застосунку для пошуку та порівняння ціни автомобілів. Проаналізовано мобільні застосунки для пошуку та порівняння ціни автомобілів: AUTO.RIA, AutoUncle, AutoScout24. Програмно реалізовано пошуку автомобілів, додавати автомобілі до списку бажаного, сортування списку бажаного за ціною для порівняння ціни автомобілів, створення власного профілю, фільтрації оголошень про продаж автомобілів за моделлю, перегляду оголошення детально, перегляду профілю продавця, створення власного оголошення про продаж автомобілів. Проведено функціональне тестування.

Сферою використання застосунку є люди, які шукають автомобіль для особистого користування.

КЛЮЧОВІ СЛОВА: KOTLIN, ANDROOID, ANDROID STUDIO, ANDROID JETPACK, FIREBASE, NOSQL, JSON.

ЗМІСТ

ВСТУП

У теперішньому часі мобільні застосунки відіграють все більше у нашому житті. Одним з невід'ємних можливостей це пошук автомобілів та порівняння їх ціни. Люди бажають знайти та придбати авто та виключно за вигідною ціною. Мобільний застосунок може зібрати все необхідне та корисне під один клік.

Тому для розробки пошуку та порівняння ціни автомобілів було обрано мобільний застосунок. Таке рішення надасть зібрати все необхідне охопивши величезну аудиторію. Таких мобільних застосунків існує багато та не всі вони відповідають потребам. Багато з них мають не повний функціонал або незручний інтерфейс.

Об'єктом дослідження є розробка мобільного застосунку для пошуку та порівняння ціни автомобілів.

Предметом дослідження є мобільний застосунок для пошуку та порівняння ціни автомобілів.

Метою роботи є спрощення процесу пошуку та порівняння автомобілів за рахунок застосування мобільного застосунку.

Методи дослідження ознайомитись з існуючими мобільними застосунками та визначити вимоги до власного застосунку. Обрати технології, за допомогою яких буде розроблений застосунок, а саме Kotlin, Android Jetpack, Firebase, Firebase Storage, Cloud Firestore NoSQL, Android Studio.

В ході створення мобільного застосунку відбувалось дослідження процесу розробки.

Наукова новизна роботи зосереджується в розробці мобільного застосунку для пошуку та порівняння ціни автомобілів з використанням новітніх технологій та сучасних архітектурних рішень для його реалізації.

Практична значущість результатів дослідження зосереджується у наданні користувачам розробленого мобільного застосунку для пошуку та порівняння ціни автомобілів. Такий додаток надасть можливість ефективно знаходити найбільш вигідні пропозиції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення мобільного застосунку

Мобільний застосунок (мобільний додаток) – це програма, яка була розроблена для планшетів, смартфонів та інших мобільних пристроїв. Багато мобільних застосунків можна знайти у крамницях застосунків, ці крамниці будуть відрізнятися залежно від операційної системи, таких як App Store на IOS, Google Play (Google Play Market) на Android, Windows Phone Store для особистої платформи Windows Phone та інші. Здебільшого мобільні застосунки у крамницях застосунків є безкоштовні та є такі, що зустрічаються за оплату.

Мобільні застосунки розробляються під різні потреби: ігри, фінанси, покупки, месенджери, навчання та багато іншого. Ціль мобільних застосунків спростити життя людей, забравши буденні довготривалі справи, замінивши їх простими діями. Таким застосунком може стати застосунок для пошуку та порівняння ціни автомобілів. Цей додаток допоможе швидко та зручно отримати доступ до всієї необхідної інформації про автомобіль.

Здебільшого застосунки для пошуку та порівняння ціни автомобілів мають часто схожі можливості:

- пошук автомобілів;
- фільтрація результатів;
- детальна інформація про автомобіль;
- збереження обраних автомобілів;
- створення власних оголошень для продажу автомобілів;
- оцінка автомобілів;
- зв’язок з продавцем;
- авторизація та профіль користувача.

Щоб почати використовувати програму для порівняння цін на автомобілі, користувачі повинні спочатку завантажити програму з певного магазину програм, наприклад Google Play або App Store, залежно від операційної системи пристрою.

Після завантаження та встановлення програми користувач повинен створити обліковий запис або увійти під наявним обліковим записом, якщо це вже було зроблено.

Обліковий запис дозволить зберігати улюблені оголошення про автомобілі, порівнювати їх, відстежувати тенденції цін і використовувати інші функції, які пропонує додаток.

Після входу в систему за допомогою свого облікового запису користувач зможе використовувати такі функції програми, як пошук транспортних засобів за різними критеріями, відображення детальної інформації про обрану модель автомобіля, вибір і порівняння їх характеристик і цін.

У детальному огляді автомобіля користувачі можуть знайти можливість зв’язатися з продавцем.

1.2 Огляд існуючих застосунків для пошуку та порівняння ціни автомобілів

Сьогодні на ринку існує багато мобільних додатків і веб-сайтів, які дозволяють користувачам шукати та порівнювати ціни на автомобілі. Для огляду було обрано три схожі мобільні додатки для пошуку та порівняння цін на автомобілі. Серед них AUTO.RIA, AutoUncle і AutoScout24.

1.2.1 AUTO.RIA

AUTO.RIA [2] – популярний український сайт з купівлі та продажу нових і вживаних автомобілів в Україні.

Окрім сайту, AUTO.RIA також пропонує мобільний додаток для iOS та Android.

Згідно з даними Google Play, додаток AUTO.RIA станом на весну 2024 року завантажили понад п’ять мільйонів разів із хорошим рейтингом огляду 4,6.

За допомогою додатка можна шукати автомобілі за різними категоріями, а саме за типом автомобіля, кузовом тип, марка, модель, рік випуску, ціна, регіон, технічні характеристики, потужність двигуна та багато іншої інформації.

Додаток дозволяє користувачам створювати власні акаунти, дозволяючи зберігати цікаві оголошення про автомобілі. Також цей застосунок має можливість розміщувати власні оголошення про продаж автомобілів.

Він також може відображати детальну інформацію про транспортний засіб.

AUTO.RIA має можливість зв'язатися з продавцем. Спілкування здійснюється шляхом відправки текстового повідомлень або дзвінка продавцю.

Недоліком є низька продуктивність старих смартфонів із низькою продуктивністю.

На рисунку 1.1 показаний екран програми AUTO.RIA.

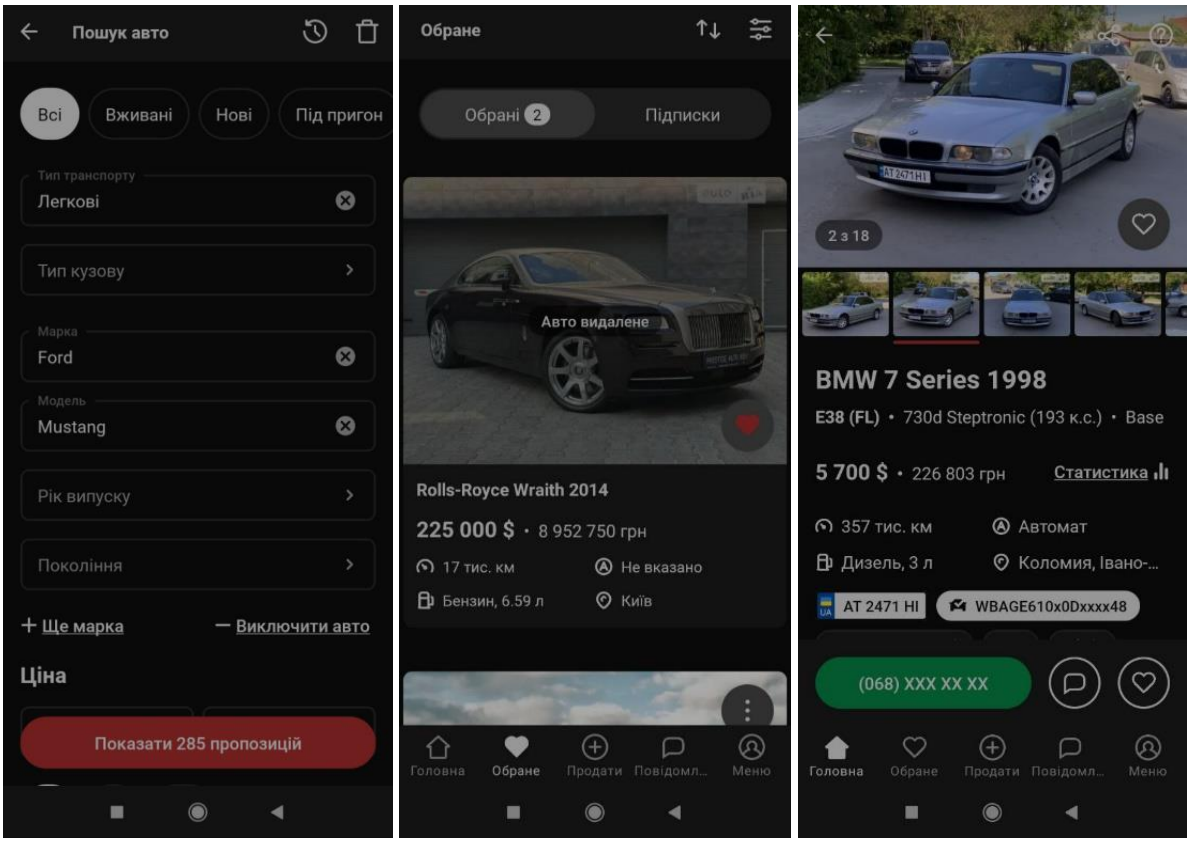


Рис. 1.1 Приклади екранних форм додатку AUTO.RIA

1.2.2 AutoUncle

AutoUncle [4] – це незалежний мобільний додаток для пошуку вживаних автомобілів, який збирає, порівнює та переглядає понад 11 мільйонів пропозицій з понад 1900 європейських сайтів, включаючи найбільші портали вживаних автомобілів і веб-сайти дилерів.

Мобільний застосунок доступний для iOS та Android.

Згідно з даними з Google Play, додаток AutoUncle станом на весну 2024 року завантажили понад один мільйон разів із хорошим рейтингом огляду 4,8.

З незвичних функцій, що відрізняє цей застосунок від інших є оцінка ціни. Вона ґрунтується на більш ніж 100 параметрах та порівнює пропозицію на ринку, щоб оцінити кожен автомобіль.

AutoUncle пропонує багато функцій, включаючи можливість пошуку вживаних автомобілів за різними критеріями, включаючи марку, модель, рік, ціну, пробіг і тип палива.

Користувач також може додавати оголошення до списку улюблених та порівнювати їх поруч.

Додаток надає можливість отримувати сповіщення про нові повідомлення, які відповідають критеріям пошуку.

Кожне оголошення містить детальний опис, зображення та технічні характеристики обраного автомобіля.

Недоліком є доступність лише в 14 європейських країнах.

На рисунку 1.2 показаний екран програми AUTO.RIA.

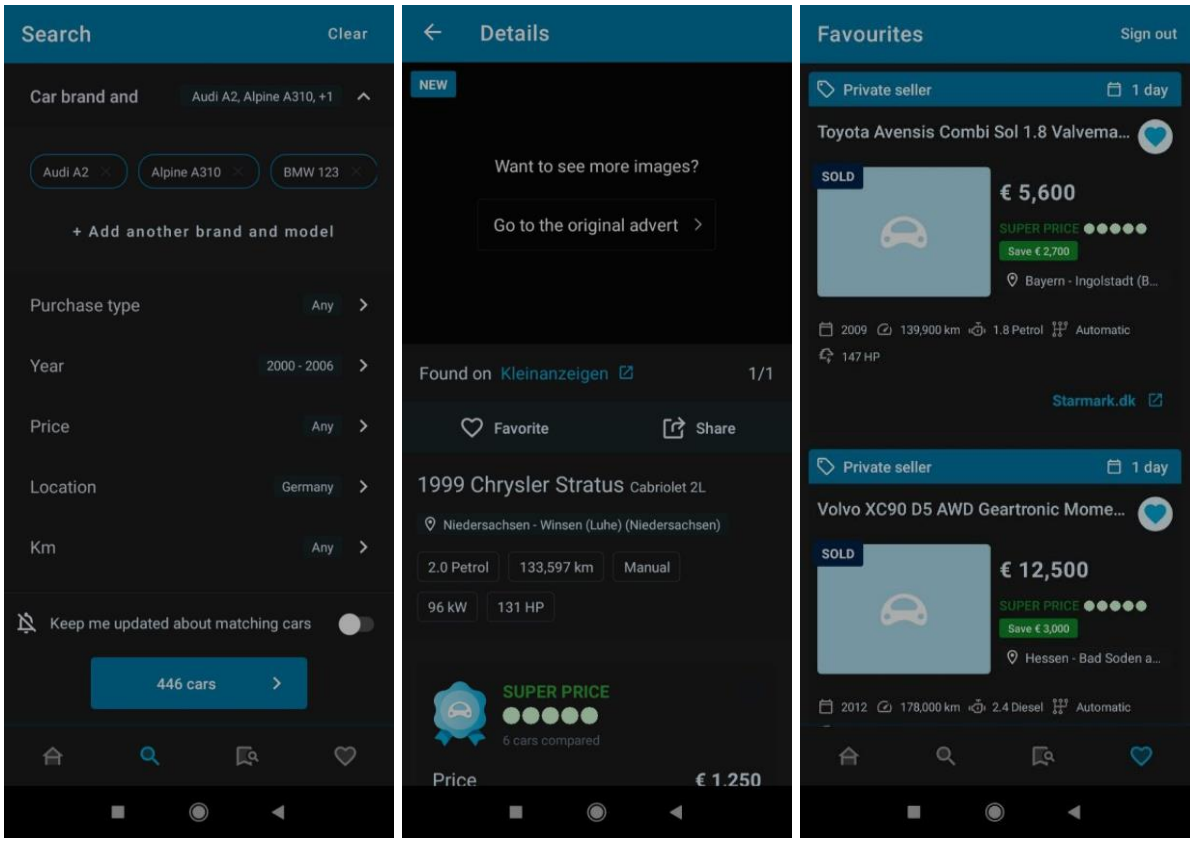


Рис. 1.2 Приклади екранних форм додатку AutoUncle

1.2.3 AutoScout24

AutoScout24 [3] – популярна онлайн-платформа для купівлі та продажу нових і вживаних автомобілів у Європі.

Також має власний мобільний застосунок для пошуку та купівлі автомобілів.

Додаток AutoScout24 доступний на пристроях iOS та Android.

За даними Google Play, станом на весну 2024 року AutoScout24 було завантажено більше десяти мільйонів разів із хорошим рейтингом 4,8.

Використовує різні фільтри, такі як марка, модель, рік випуску, пробіг, тип палива тощо, що дозволить знайти правильний автомобіль для потреб поза звичайним пошуком.

Цей додаток дозволяє порівнювати різні автомобілі за технічними характеристиками, ціною та іншими параметрами.

Збереження автомобільних оголошень, для перегляду пізніше.

Створення оголошень для продажу, після чого автомобіль буде розміщено на веб-сайті та в додатку AutoScout24.

Зв’язок з продавцем безпосередньо через додаток.

Мінусом є те, що він доступний лише в європейських країнах. Також користувачі жаліються на повільну працю додатку. Присутня реклама.

На рисунку 1.3 показаний екран програми AutoScout24.

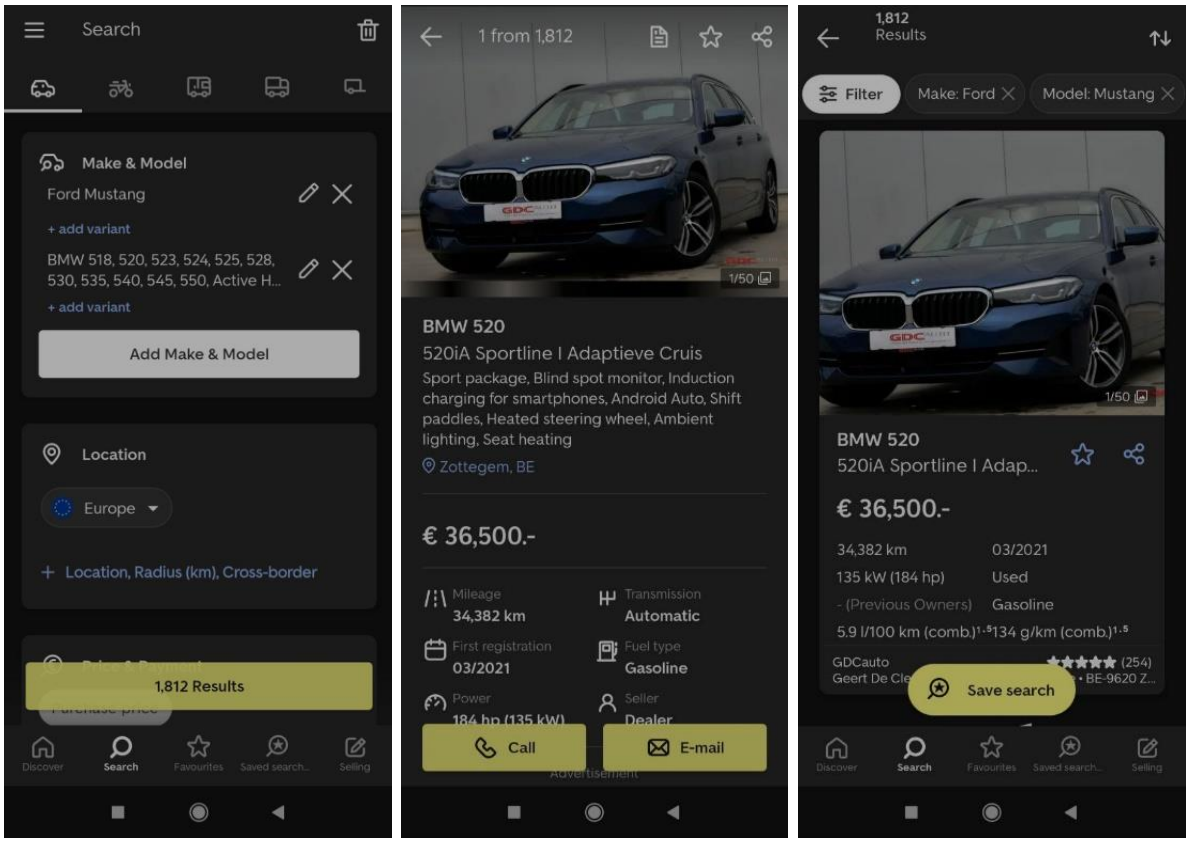


Рис. 1.3 Приклади екранних форм додатку AutoScout24

На таблиці 1.1 наведено результати аналізу розглянутих мобільних застосунків для пошуку та порівняння ціни автомобілів.

Таблиця 1.1

Результати аналізу мобільних застосунків

Функціонал	AUTO.RIA	AutoUncle	AutoScout24
Розширений пошук	+	+	+
Порівняння автомобілів	-	-	-
Розміщення оголошення про продаж авто	+	-	+
Додавання до списку вподобаного	+	+	+
Створення власного аккаунту	+	+	+
Редагування акаунту	+	-	+

1.3 Висновки аналізу

Після розуміння концепції мобільних застосунків та огляду існуючих застосунків можна вказати на переваги та недоліки застосунків пошуку автомобілів та порівняння цін, вони наведені в таблиці 1.2.

Таблиця 1.2

Переваги та недоліки мобільних застосунків

Переваги	Недоліки
Пошук автомобілів	Реклама
Збереження оголошень	Повільність застосунку
Можливість створення оголошень	Регіональна доступність
Зв'язок з продавцем	
Обліковий запис	

2 ЗАСОБИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Android Studio [7] – це інтегроване середовище розробки (IDE) для операційної системи Google Android, створене на основі програмного забезпечення IntelliJ IDEA від JetBrains і розроблене спеціально для розробки на Android.

Android Studio має емулятор, тобто програмне забезпечення, яке імітує фізичний пристрій Android на комп'ютері. Це дозволяє тестувати програми на різних віртуальних пристроях, не потребуючи кожного фізичного телефону чи планшету. Емулятор може надати майже всі функції фізичного пристрою на Android. Він може імітувати датчики, як акселерометр та GPS, також має можливість здійснювати дзвінки та надсилати текстові повідомлення (у віртуальному середовище). Коли вносяться зміни у програму не потрібно переносити її на фізичний пристрій, запуск програми на емуляторі є швидшим варіантом. Легко налаштовувати віртуальний пристрій, наприклад заряд акумулятора або потужність сигналу Wi-Fi.

Кожен проект у Android Studio має один або декілька модулів із файлами вихідного коду та файлами ресурсів. Модуль - це логічна одиниця коду, яку можна використовувати для покращення структури, повторного використання коду та залежностей.

При створенні нового проекту Android Studio створює структуру за замовчуванням у цю структуру входять наступні папки:

– папка «app» містить код застосунку, написаний на мові програмування Kotlin або Java. Також включає ресурси програми, макети інтерфейсу, зображення та файли конфігурації;

– папка «manifests» містить у собі важливий файл «AndroidManifest.xml», який містить метадані про застосунок: назва, версія, компоненти та дозволи які потребує додаток;

– папка «kotlin+java» зберігає вихідний код проекту, написаний на Kotlin або на Java.

– папка «res» зберігає усі ресурси застосунку, які не є вихідним кодом. Перелік програм які зберігаються у папці «res»: «drawable», «layout», «mipmap», «values»;

– папка «drawable» зберігає зображення, які використовуються у застосунку;

– папка «layout» потрібна для зберігання XML-файлів, які відповідають за графічну візуалізацію структури застосунку;

– папка «mipmap» знаходяться піктограми для застосунку різних розмірів та роздільної здатності;

– папка «values» містить файли конфігурації кольорів, стилів, рядків, тем.

Структура проекту за замовчуванням наведена рисунку 2.1

Android Studio використовує Gradle як основу для побудови системи. Gradle є інструментом автоматизації збірки для розроблення програмного забезпечення. Він керує процесом розробки від компіляції до тестування. Ключовими особливостями є підтримка багатьох мов включаючи Java та Kotlin. Gradle керує залежностями бібліотек проекту, даючи можливість завжди використовувати правильну версію бібліотеки. Кожен проект Android Studio містить папку «gradle» у якій містяться файли сценаріїв Grandle. Ці файли визначають, як збирається застосунок.

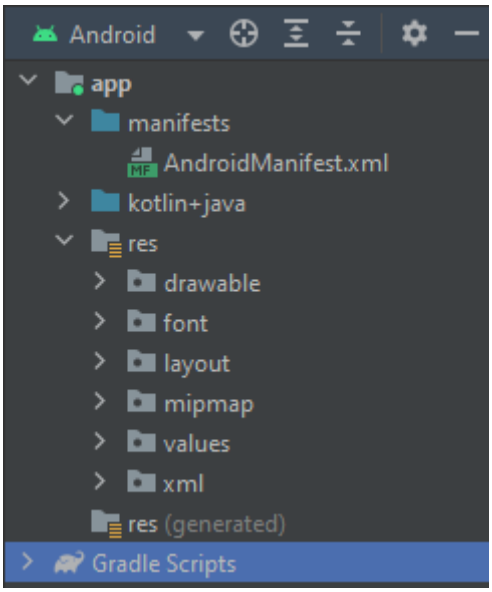


Рис. 2.1 Структура проекту за замовчуванням у Android Studio

2.2 Мова програмування

2.2.1 Kotlin

Kotlin [6] – це статично типізована мова програмування, яка працює на віртуальній машині Java (JVM). Розроблена компанією JetBrains для полегшення розробки програмного забезпечення. Kotlin є офіційною мовою програмування розробки застосунків для Android.

Синтаксис Kotlin відрізняється від інших мов програмування більш полегшеною для програміста читабельністю завдяки написання менше коду, що не є необхідним для функціонування. Загалом, стисліть Kotlin робить написання великих проектів більш керованим, оскільки написання одних і тих самих функцій часто вимагає менше рядків програмування, ніж Java. Крім того, він знає, як бути коротким і по суті без шкоди для читабельності синтаксису.

На початку появи мобільних застосунків на Android мовою програмування була Java. Але з часом її місце посіла мова програмування Kotlin через ряд переваг:

– у Java можна мати null значення, призначеній змінній, що може призвести до винятків null pointer під час виконання. Kotlin, з іншого боку,

вимагає явного визначення, чи може змінна бути null чи ні. Це полегшує уникнення винятків null pointer під час виконання;

– Kotlin дозволяє розширити функціональність існуючих класів, не змінюючи їх код. Це дозволяє зменшити кількість користувацьких методів утиліт і спрощує розробку;

– у Kotlin існують спеціальні класи для представлення даних, які називаються класами даних. Вони автоматично генерують toString(), Equals(), hashCode() та інші методи, що полегшує роботу з даними;

– Kotlin підтримує багато функціональних концепцій, таких як лямбда-вирази, функції високого рівня, замикання тощо, що робить код легшим для розуміння та більш компактним;

– Kotlin автоматично розпізнає типи даних в умовних виразах, якщо вони були перевірені вище у коді, зменшуючи кількість вимог до перевірки типів;

– Kotlin підтримує coroutines (співпрограми), що дозволяє легко працювати з асинхронним кодом, що робить його легшим для розуміння та більш ефективним.

2.2.2 XML

XML (Extensible Markup Language) [11] – це мова розмітки, яка використовується для структурування та обміну інформацією між різними системами. У XML не можна виконувати обчислювальні операції самостійно. Зате для керування структурованими даними можна застосовувати будь-яку мову програмування чи програмне забезпечення.

В Android Studio XML використовується для опису макета інтерфейсу користувача програми Android. Файли XML визначають такі елементи інтерфейсу користувача, як текстові поля, кнопки та зображення, а також їхні позиції та властивості.

Перевага використання XML для інтерфейсу користувача полягає в тому, що він описує, як має виглядати інтерфейс користувача, а не те, як його створити програмним шляхом.

XML також дозволяє легко налаштовувати макети для різних екранів мобільних пристроїв і роздільної здатності. Крім того, він відокремлює логіку програми від її представлення, що полегшує розуміння та керування кодом.

Android Studio надає візуальний редактор XML-макетів, щоб полегшити створення макетів інтерфейсу користувача. Крім того, не втрачається можливість створювати XML-макети вручну за допомогою текстового редактора.

Одними з основних елементів XML для інтерфейсу користувача є:

1. <LinearLayout> – розміщує дочірні компоненти один за одним у горизонтальному або вертикальному напрямку.
2. <RelativeLayout> – розміщує дочірні компоненти відносно один одного або на межах макета.
3. <TextView> – відображає текстовий вміст.
4. <Button> – кнопка для виконування дії, яка буде визначена.
5. <ImageView> – відображає зображення.
6. <EditText> – поле вводу, користувач може вводити свій текст.

Також у XML є атрибути, які використовуються для налаштування властивостей компонентів інтерфейсу користувача. Наприклад, використання атрибуту «text», щоб визначити текст кнопки, або атрибут «src», щоб визначити джерело зображення.

2.3 Набір інструментів Android Jetpack

Android Jetpack [1] – це набір бібліотек, інструментів та рекомендацій, що допомагають розробникам писати швидко та легко застосунки для Android.

Бібліотеки Android Jetpack знаходяться в просторі імені androidx.

Jetpack Compose є частиною Android Jetpack це декларативна структура інтерфейсу користувача з відкритим кодом для Android, створена за допомогою Kotlin від Google. Це дозволяє створювати користувацькі інтерфейси у більш стислий та інтуїтивно зрозумілий спосіб порівняно з традиційними підходами. Замість того, щоб писати імперативний код для оновлення інтерфейсу

користувача, описується бажаний стан інтерфейсу користувача, і Jetpack Compose автоматично перекомпонує інтерфейс користувача відповідно до змін. Це призводить до чистішого та легшого для обслуговування коду. Compose використовує парадигму реактивного програмування, автоматично оновлюючи елементи інтерфейсу користувача у відповідь на зміни в даних або стані. Компоненти інтерфейсу користувача в Compose визначаються як складові функції, які є легкими, модульними та придатними для повторного використання. Компоновані функції можуть приймати параметри та повертати елементи інтерфейсу користувача, що дозволяє легко створювати складні інтерфейси користувача з менших будівельних блоків.

Jetpack Navigation відноситься до бібліотек Android Jetpack. Ця бібліотека спрощує процес керування навігацією між різними пунктами призначення у застосунку Android.

В основі компонента Navigation лежить NavGraph, який представляє навігаційну структуру застосунку. Він визначає зв'язки між різними місцями призначення та діями, які викликають навігаційні переходи. Місця призначення – це окремі екрани або елементи інтерфейсу в додатку, до яких користувачі можуть перейти. Дії визначають можливі переходи між пунктами призначення в межах NavGraph. Ці переходи можуть бути викликані взаємодією користувача, наприклад, натисканням кнопки або вибором елемента зі списку. NavHost - це контейнер у інтерфейсі програми, де відображаються місця призначення. Він зазвичай реалізується як FragmentContainerView або NavHostFragment. SafeArgs допомагає запобігти помилкам під час виконання, гарантуючи, що аргументи передаються правильно і з правильними типами даних. Navigation підтримує вбудовані анімації для переходів між місцями призначення, що дозволяє легко створювати візуально привабливі навігаційні враження.

WorkManager – це бібліотека в пакеті Android Jetpack, він зосереджений на управлінні фоновими завданнями у застосунках Android. Його призначення забезпечувати безпечне виконання важливих фонових завдань, навіть коли застосунок закривається або пристрій перезавантажується.

WorkManager автоматично вибирає найбільш підходящий метод виконання фонових завдань на основі рівня API пристрою, забезпечуючи зворотну сумісність для пристроїв із версіями Android аж до рівня API 14. Він пропонує, як виконуються завдання, дозволяючи вказувати обмеження, такі як доступність мережі, стан заряду та стан очікування пристрою, щоб оптимізувати виконання роботи. WorkManager надає багато функцій спостереження, що дозволяє відстежувати стан завдань, включаючи їх хід, завершення та невдачу, за допомогою LiveData.

LiveData – це компонент Android Jetpack, який використовується для управління даними, які можуть змінюватися з часом. Він спрощує роботу з непостійними даними, такими як дані з бази даних або дані, отримані з мережі, та автоматично оновлює інтерфейс користувача, коли ці дані змінюються.

LiveData дотримується життєвого циклу інших компонентів застосунку, таких як дії, фрагменти або служби. Він оновлює компоненти інтерфейсу користувача, які знаходяться в активному стані життєвого циклу, запобігаючи можливим витокам пам'яті та зменшуючи кількість помилок через null посилання.

ViewModel є ключовим компонентом в Android Jetpack, розробленим для управління та зберігання даних, пов'язаних з інтерфейсом користувача, з урахуванням життєвого циклу. Це дозволяє даним не оновлюватись після змін конфігурації, таких як повороти екрану.

ViewModel допомагає відокремити дані інтерфейсу та бізнес-логіку від контролерів інтерфейсу, таких як Activity та Fragment. Це призводить до чистішої та більш модульної архітектури. Оскільки ViewModel не оновлює зміни конфігурації, він може зберігати дані, пов'язані з інтерфейсом, які можуть бути швидко використані контролерами інтерфейсу, зменшуючи необхідність повторного завантаження або перерахунку даних. Одним з найпотужніших поєднань є використання ViewModel з LiveData. Коли дані в LiveData змінюються, він автоматично оновлює будь-які пов'язані компоненти

інтерфейсу, які за ними спостерігають, забезпечуючи синхронізацію інтерфейсу з даними.

ConstraintLayout – це менеджер макету в Android Jetpack, який дозволяє створювати складні макети з плоскою ієрархією представлень. Його було введено для забезпечення більшої гнучкості та покращення продуктивності у порівнянні з традиційними макетами, такими як LinearLayout та RelativeLayout. ConstraintLayout особливо корисний для розробки адаптивних інтерфейсів користувача, які можуть адаптуватися до різних розмірів екранів та орієнтацій.

AppCompat є частиною Android Jetpack, забезпечує зворотну сумісність версій компонентів Android, дозволяючи використовувати найновіші функції на старіших версіях Android. AppCompat надає підтримку бібліотек, які дозволяють використовувати функції новіших версій Android на старіших версіях, починаючи з Android 2.1, хоча новіші версії AppCompat вимагають вищого мінімального API рівня.

Android KTX (Kotlin Extensions) – це набір розширень Kotlin, наданих Google, які покращують API фреймворку Android. KTX надає більш плавну поверхню API, схожу на Kotlin, для типових завдань Android, зменшуючи шаблонний код і покращуючи читабельність.

CameraX є частиною Android Jetpack, розробленою для спрощення процесу інтеграції функцій камери у застосунки Android. Надає простий у використанні API для доступу до камери на різних пристроях Android, незалежно від основного апаратного забезпечення чи версії Android.

Бібліотека Media в Android Jetpack надає основні функції для відтворення та запису медіафайлів у застосунках Android. Ця бібліотека спрощує взаємодію мультимедійних функцій без необхідності мати справу зі складністю низькорівневих API.

ExoPlayer – це бібліотека медіаплеєрів, розроблена Google. Він підтримує різні мультимедійні формати та протоколи адаптивної потокової передачі, такі як DASH (динамічна адаптивна потокова передача через HTTP) і HLS (пряма трансляція HTTP). ExoPlayer легко настроюється та надає розширені функції, як

підтримку DRM (Digital Rights Management) та повну взаємодію з іншими бібліотеками Jetpack.

MediaSessionCompat надає уніфікований інтерфейс для взаємодії з сеансами відтворення медіа, дозволяючи керувати відтворенням з різних джерел. MediaSessionCompat також взаємодіє системними компонентами, як сповіщення, дозволяючи керувати відтворенням, навіть коли програма працює у фоновому режимі.

MediaBrowserServiceCompat – дозволяє застосункам переглядати вміст медіа, наданий застосунком та запросити застосунок відтворити вміст медіа. Може використовуватись для керуванням вмістом, який відтворюється за допомогою MediaSessionCompat.

NotificationManager – службовий сервіс керує всіма сповіщеннями на пристрої. Він відповідає за відображення сповіщень користувачу, обробку взаємодії користувача зі сповіщеннями та скасування сповіщень, коли вони більше не потрібні.

NotificationCompat – частина бібліотеки AndroidX забезпечує сумісність для створення сповіщень, які сумісні з широким спектром версій Android, забезпечуючи використання на різних пристроях.

2.4 Набір інструментів Firebase для реалізації серверної частини

Для розробки серверної частини буде використовуватись Firebase. Firebase [5] – це платформа, що надає набір інструментів для розробки веб та мобільних застосунків. На рисунку 2.2 наведено різниця між звичайними серверами та Firebase.

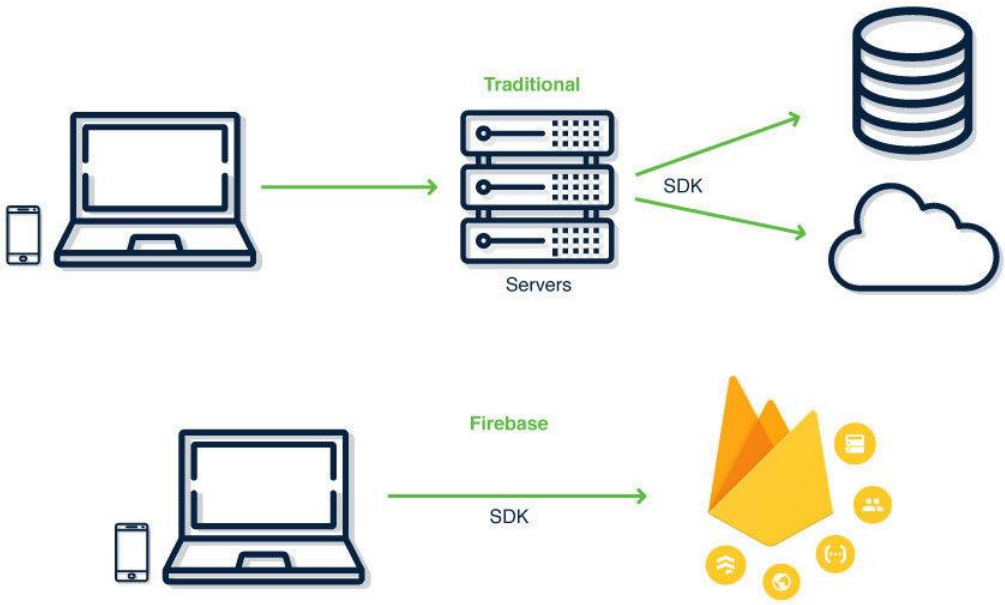


Рис. 2.2 Порівняння між звичайними серверами та Firebase

Firebase Realtime Database – це хмарне сховище даних, яке зберігає інформацію у форматі JSON. Особливістю цієї бази є те, що дані синхронізуються в режимі реального часу з усіма підключеними клієнтами. Це означає, що коли створюються програми для різних платформ, усі користувачі працюватимуть з тією самою версією Realtime Database і автоматично отримуватимуть оновлені дані.

Firebase Realtime Database дає змогу створювати повноцінні спільні програми, надаючи безпечний доступ до бази даних безпосередньо з коду користувача. Дані зберігаються локально, що дозволяє продовжувати отримувати події в реальному часі навіть у режимі офлайн, забезпечуючи безперебійну та оперативну роботу. Після відновлення з'єднання база даних реального часу синхронізує локальні зміни з віддаленими оновленнями, які виконуються в автономному режимі, автоматично вирішуючи конфлікти.

Realtime Database використовує гнучку мову правил безпеки під назвою Firebase Realtime Database Security Rules, яка визначає структури даних і дозволяє контролювати, коли та як можна читати чи записувати формат даних.

У поєднанні з автентифікацією Firebase розробники можуть точно визначити, хто має доступ до певних даних і як.

Realtime Database – це база даних NoSQL, яка надає різноманітні функції та оптимізації порівняно з реляційними базами даних. API бази даних реального часу розроблено спеціально для швидкої роботи.

NoSQL [8] – це тип бази даних, яка використовує методи зберігання та отримання даних, які відрізняються від традиційного реляційного підходу використання таблиць і зв'язків.

Бази даних NoSQL часто використовуються для зберігання великих обсягів неструктурованих або слабко структурованих даних, таких як текстові файли, зображення та відео.

Ключові особливості баз даних NoSQL:

– NoSQL дозволяє зберігати різні типи даних без попереднього визначення схеми таблиці;

– Багато баз даних NoSQL розроблено для легкого горизонтального масштабування, що означає додавання нових серверів для обробки більшої кількості даних;

– Загалом NoSQL добре підходить для завдань, які потребують високошвидкісних операцій запису та читання.

Firebase Authentication надає послуги на стороні серверу, прості у використанні SDK і готові до використання бібліотеки інтерфейсу користувача для автентифікації користувачів у програмі. Цей інструмент підтримує автентифікацію за допомогою паролів, номерів телефонів і таких платформ, як Google, Facebook, Twitter тощо.

Щоб авторизувати користувача в програмі, потрібно спочатку отримати його облікові дані. Наприклад, це може бути електронна адреса та пароль або токен OAuth із соціальної мережі. Потім ці дані передаються до Firebase Authentication SDK. Служба сервера перевіряє ці дані та надсилає відповідь користувачеві.

Після успішної автентифікації можна отримати доступ до основної інформації профілю користувача та контролювати їхній доступ до даних, що зберігаються в інших продуктах Firebase. Крім того, виданий токен автентифікації можна використовувати для автентифікації користувачів у службах сервера.

Cloud Storage – це служба зберігання об'єктів, розроблена Google для масштабування. Використовуючи Firebase SDK для хмарного сховища, можна додати засоби захисту Google для завантаження файлів для додатків на основі Firebase незалежно від умов мережі.

Клієнтський SDK дозволяє зберігати зображення, аудіо, відео та інший спеціальний вміст. На стороні сервера є можливість використовувати Firebase Admin SDK для керування фрагментами, створення URL-адрес для завантаження та використання API Google Cloud Storage для доступу до файлів.

Cloud Functions – це безсерверний фреймворк, який дозволяє автоматично запускати серверний код у відповідь на різні події. Це можуть бути фонові операції, запити HTTPS, завдання за допомогою Admin SDK або Cloud Scheduler. Код, написаний на JavaScript, TypeScript або Python, зберігається та виконується в інфраструктурі Google Cloud. Це означає, що не потрібно турбуватися про керування та масштабування власних серверів, оскільки все виконується в керованому середовищі.

Щойно функція створена та розгорнута, Google одразу бере на себе керування нею. Функцію можна запустити безпосередньо HTTP-запитом, Admin SDK, запланованим завданням або, у випадку фонових функцій, сервери Google очікують на подію та запускають функцію, коли вона запускається.

Зі збільшенням навантаження Google швидко збільшує кількість екземплярів віртуального сервера, необхідних для запуску функції. Кожна функція працює автономно, в окремому середовищі зі своїми налаштуваннями.

Firebase Cloud Messaging (FCM) — це популярне міжплатформне рішення для обміну повідомленнями, яке дозволяє надсилати повідомлення безкоштовно та безпечно. За допомогою FCM можна повідомити користувача, коли нові дані

доступні для синхронізації. Також є можливість надсилати сповіщення повторно. Для таких сценаріїв, як обмін миттєвими повідомленнями, FCM дозволяє передавати користувачу дані розміром до 4 кілобайт.

Firebase Remote Config — це хмарна служба, яка дозволяє змінювати функціональність і зовнішній вигляд клієнтської або серверної програми без необхідності оновлення програми користувачеві. Під час роботи з Remote Configuration встановлюються стандартні параметри програми, які визначають поведінку та зовнішній вигляд програми. Використовуючи консоль Firebase або API сервера Remote Config, ці параметри за замовчуванням можна пізніше змінити для всіх користувачів або для окремих сегментів бази користувачів. Програма або сервер вирішує, коли застосовувати ці оновлення, регулярно перевіряючи наявність нових налаштувань і розгортаючи їх з мінімальним впливом на продуктивність.

Firebase Test Lab – це хмарна платформа тестування додатків, яка дає змогу тестувати додаток на різних пристроях і конфігураціях. Це дає змогу краще зрозуміти, як працюватиме програма в реальному середовищі, коли її використовуватимуть реальні користувачі.

Test Lab використовує реальні пристрої, які працюють у центрах обробки даних Google, щоб перевірити програму. Ці пристрої мають оновлені API та змінені параметри локалізації, що дозволяє перевірити програму на апаратному рівні та параметрами, з якими вона зіткнеться в реальному світі.

Firebase Crashlytics — це інструмент, який автоматично виявляє та повідомляє про збої програми в режимі реального часу. Допомагає визначити проблеми та вирішити їх. Crashlytics прискорює процес налагодження, логічно групуючи збої та надаючи пояснення, щоб краще зрозуміти, чому вони сталися.

Firebase Hosting — це платформа для хостингу веб-сайтів, розроблена для розробників. Ця послуга дозволяє швидко розгортати веб-додатки в глобальній мережі CDN за допомогою лише однієї команди. Firebase Hosting розроблено в основному для статичних і односторінкових веб-додатків, його можна поєднати

з Cloud Functions або Cloud Run для створення та розміщення динамічного вмісту та мікросервісів у Firebase.

2.5 Система контролю версій Git

Git [9] – це система, яка дозволяє програмістам легко співпрацювати над різними проектами. Наприклад, код для додатку може бути розроблений одночасно десятьма програмістами, які працюють з різних місць. Вони можуть вносити зміни та додавати нові функції в будь-який час незалежно один від одного. Git зберігає всі версії файлів, дозволяючи швидко повернутися до будь-якого попереднього стану. Особливості Git полягають у можливості додавання нових розробників до проекту; збереження всієї інформації, що унеможливлює втрату файлів; зміни можуть не вплинути на загальний проект, якщо вони були невдалими.

Git дозволяє використовувати гілки, які є окремими гілками різних версій одного проекту. Це корисно для введення нових функцій без впливу на основну логіку проекту. Git має одну головну гілку, та можна створювати необмежену кількість додаткових гілок. Команда gitcheck -b branch_name створює нову гілку, а щоб видалити її, використовується команда git Branch -d branch_name. Повернутися до роботи над гілкою master можна за допомогою команди gitcheck master. Щоб зробити нову гілку доступною для інших розробників, потрібно використати команду git push Origin Branch_name.

3 РОЗРОБКА ТА ОГЛЯД МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Вимоги до мобільного застосунку

Аналіз особливостей процесу пошуку та порівняння ціни автомобілів та існуючих засобів для пошуку та порівняння ціни автомобілів дозволяє сформувати вимоги до майбутнього програмного продукту. Програма пошуку та порівняння ціни автомобілів повинна бути реалізована для мобільної платформи та забезпечувати наступні функції:

– можливість пошуку автомобілів;

– можливість додавати автомобілі до списку бажаного;

– можливість сортування списку бажаного за ціною для порівняння ціни автомобілів;

– можливість створення власного профілю;

– можливість фільтрації оголошень про продаж автомобілів за моделлю;

– можливість перегляду оголошення детально;

– можливість перегляду профілю продавця;

– створення власного оголошення про продаж автомобілів.

Мобільний застосунок для пошук та порівняння ціни автомобілів матиме назву AutoFinder.

На рисунку 3.1 продемонстровано mind map до функціонування застосунку AutoFinder, що розробляється.

Mind map – це діаграма, яка використовується для візуального структурування інформації в ієрархічному порядку, що показує зв’язок між елементами цілого. Зазвичай вона створюється навколо головної ідеї, розташованої в центрі чистого аркуша паперу у вигляді зображення. До цієї центральної концепції додаються пов’язані думки, включаючи зображення, слова та фрагменти слів. Основні ідеї безпосередньо стосуються центральної теми, тоді як інші ідеї випливають із цих основних ідей.

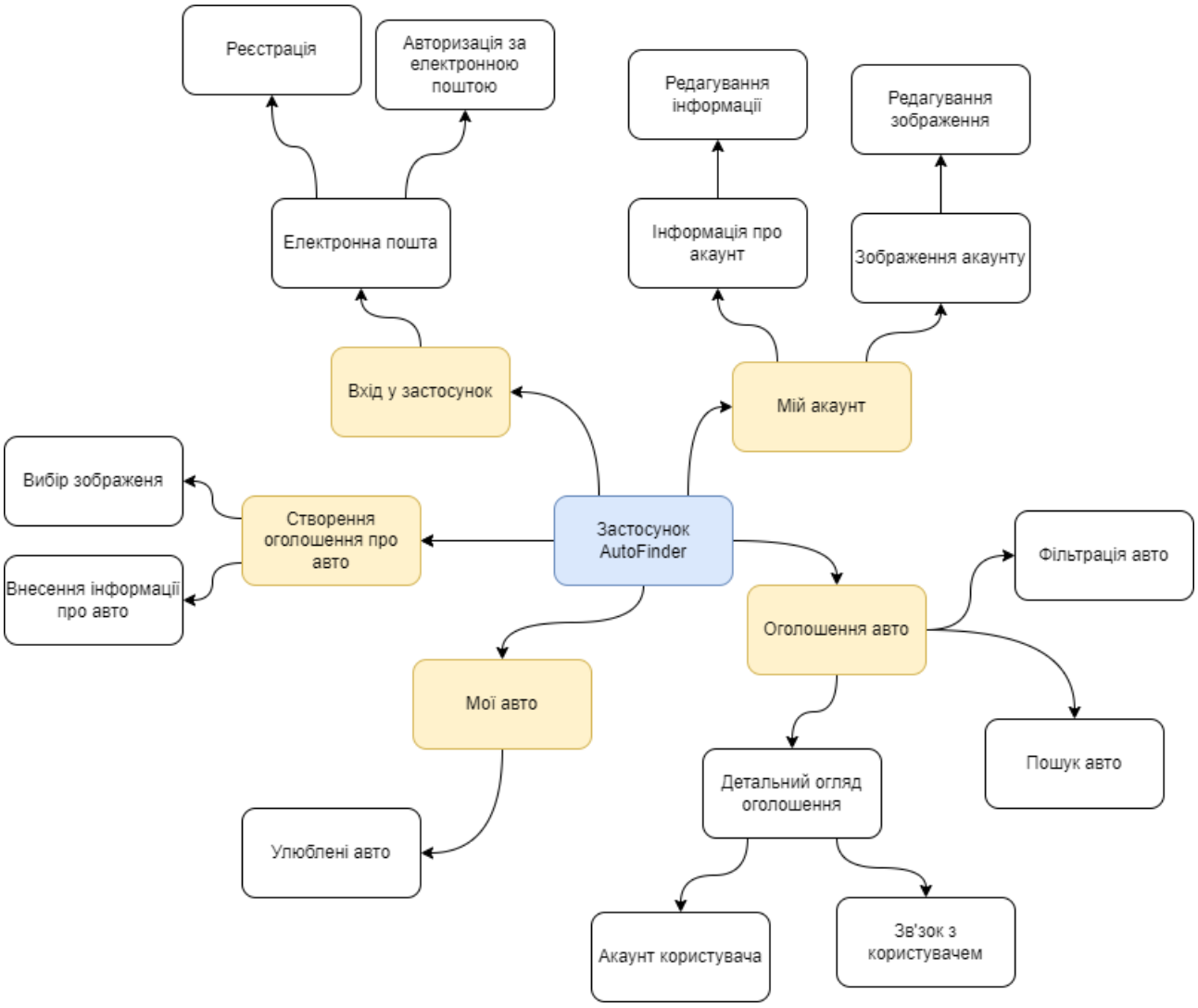


Рис. 3.1 Mind map застосунку AutoFinder

3.2 Діаграма варіантів використання

Тепер побудуємо діаграму варіантів використання. Діаграма варіантів використання ілюструє взаємодію між акторами та варіантами використання в системі. Ця діаграма показує, як різні користувачі (або інші системи) взаємодіють з певними функціями або процесами в системі. На рисунку 3.2 зображено діаграму варіантів використання.

Рис. 3.2 Діаграма варіантів використання застосунку AutoFinder

3.3 Архітектура MVVM

Для мобільного застосунку AutoFinder було обрано архітектурний патерн MVVM. Model-View-Viewmodel (MVVM) [10] – це архітектурний шаблон у розробці програмного забезпечення, який відокремлює графічний інтерфейс користувача (GUI) від бізнес або внутрішньої логіки (моделі), яка може бути реалізована за допомогою мов розмітки або коду GUI. Це гарантує, що представлення не залежить від будь-якої конкретної реалізації моделі, що полегшує розробку та обслуговування застосунку. Приклад шаблону MVVM представлено на рисунку 3.3.

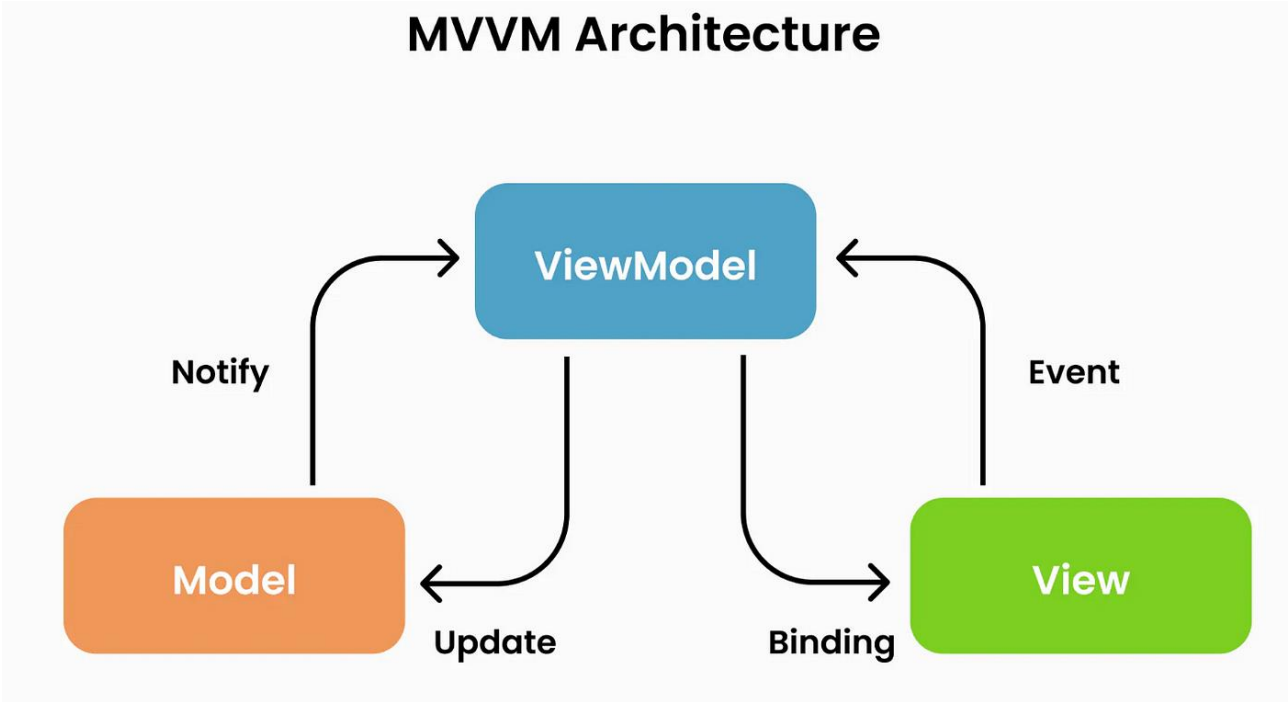


Рис. 3.3 Приклад шаблону MVVM

Model можна розуміти як модель предметної області, яка представляє фактичний стан вмісту (об’єктно-орієнтований підхід), або рівень доступу до даних, який представляє вміст (підхід, орієнтований на дані).

Подібно до шаблонів Model-View-Controller і Model-View-Presenter, View представляє структуру, макет і зовнішній вигляд того, що користувач бачить на екрані. Надає візуальне представлення моделі та керує взаємодією користувача з інтерфейсом, наприклад клацанням миші, введенням з клавіатури та жестами на екрані. Потім View передає ці взаємодії ModelView через прив’язки даних (властивості, події тощо), створюючи зв’язок між ModelView та View.

ModelView надає користувачеві загальнодоступні властивості та команди. Порівняно з контролером у шаблоні MVC і Presenter у MVP, MVVM має посередника, який автоматизує зв’язок між View і властивостями ModelView. Описує стан даних у Model. Важливою відмінністю між ModelView та Presenter у MVP є те, що Presenter має посилання на перегляд, а ModelView – ні. Замість цього View безпосередньо взаємодіє з властивостями ModelView для надсилання та отримання оновлень.

3.4 Структура класів мобільного застосунку

Класи мобільного застосунку представлені діаграмою класів.

Діаграма класів в UML є статичним представленням структури моделі. Вона відображає статичні компоненти, такі як класи, типи даних, їхній вміст і зв’язки між ними. Пакети, включаючи вкладені пакети, а також деякі поведінкові елементи можуть бути позначені на діаграмі класів.

На рисунку 3.4 наведені класи мобільного застосунку, а саме класи:

– ModelCarAd – модель оголошення про автомобіль, містить дані про автомобіль;

– FilterCarAd – клас, що відповідає за фільтрацію оголошень про автомобілі;

– AdapterCarAd – адаптер для відображення списку оголошень про автомобілі;

– LoginEmailActivity – активність для авторизації користувача за допомогою електронної пошти;

– MainActivity – головна активність додатку, яка містить основні функції навігації;

– MyAccountFragment – фрагмент, який відображає інформацію про обліковий запис користувача;

– HomeFragment – головний фрагмент, який відображає домашню сторінку додатку;

– MyCarAdsCarAdsFragment – фрагмент, який відображає оголошення користувача про автомобілі;

– MyCarAdsFavFragment – фрагмент, який відображає вибрані оголошення користувача;

– ModelImagePicked – модель вибраного зображення;

– AdapterImagePicked – адаптер для відображення вибраних зображень;

– ModelImageSlider – модель для відображення зображень в слайдері;

– CarAdDetailsActivity – активність для відображення деталей оголошення про автомобіль;

– AdCarCreateActivity – активність для створення оголошення про автомобіль;

– AdapterImageSlider – адаптер для слайдера зображень;

– CarSellerProfileActivity – активність для відображення профілю продавця автомобілів;

– LoginOptionsActivity – активність для вибору опцій авторизації;

– AdapterModels – адаптер для відображення моделей;

– Utils – клас, що містить різні корисні функції;

– ProfileEditActivity – активність для редагування профілю користувача;

– ModelCarModels – модель для відображення моделей автомобілів;

– RegisterEmailActivity – активність для реєстрації користувача за допомогою електронної пошти;

– MyCarsFragment – фрагмент, який відображає автомобілі користувача.

– AdapterCompareCar – клас для адаптації і відображення порівняння автомобілів.

– CarAdsComparisonFragment – фрагмент для відображення і порівняння оголошень про автомобілі.

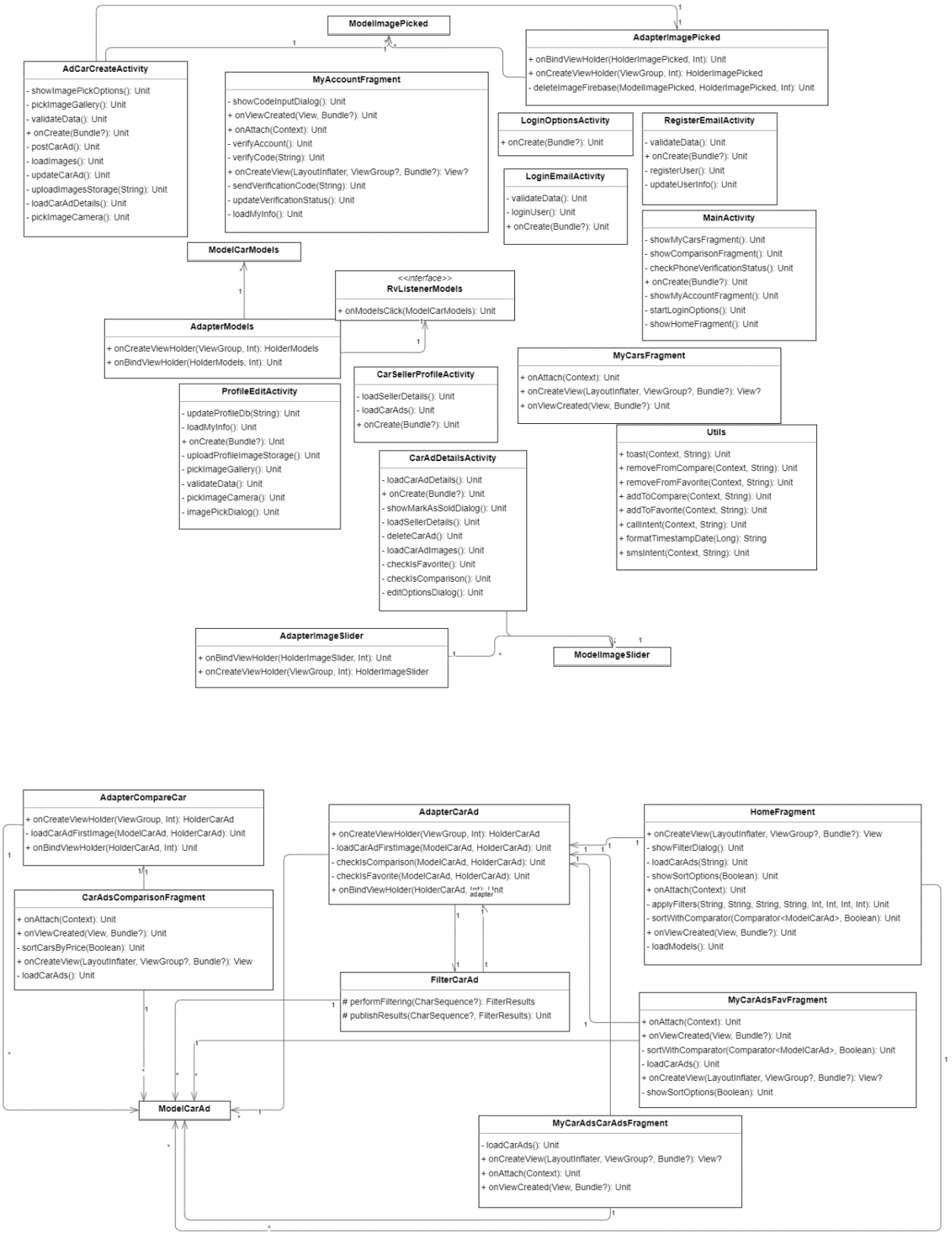


Рис. 3.4 Класи мобільного застосунку AutoFinder

3.5 Структура бази даних

Структура бази даних складається з двох основних розділів CarAds та Users. На рисунку 3.5 продемонстровано структуру бази даних у форматі JSON. Розділ CarAds містить такі поля:

– унікальний ідентифікатор;
– марка автомобіля;
– модель автомобіля;
– рік випуску автомобіля;
– локація автомобіля;
– стан автомобіля;
– опис оголошення;
– ціна автомобіля;
– статус оголошення;
– назва оголошення;
– час створення оголошення у форматі UNIX;
– ідентифікатор користувача, який розмістив оголошення;
– список зображень автомобіля, де кожне зображення має унікальний ідентифікатор та URL-адресу.

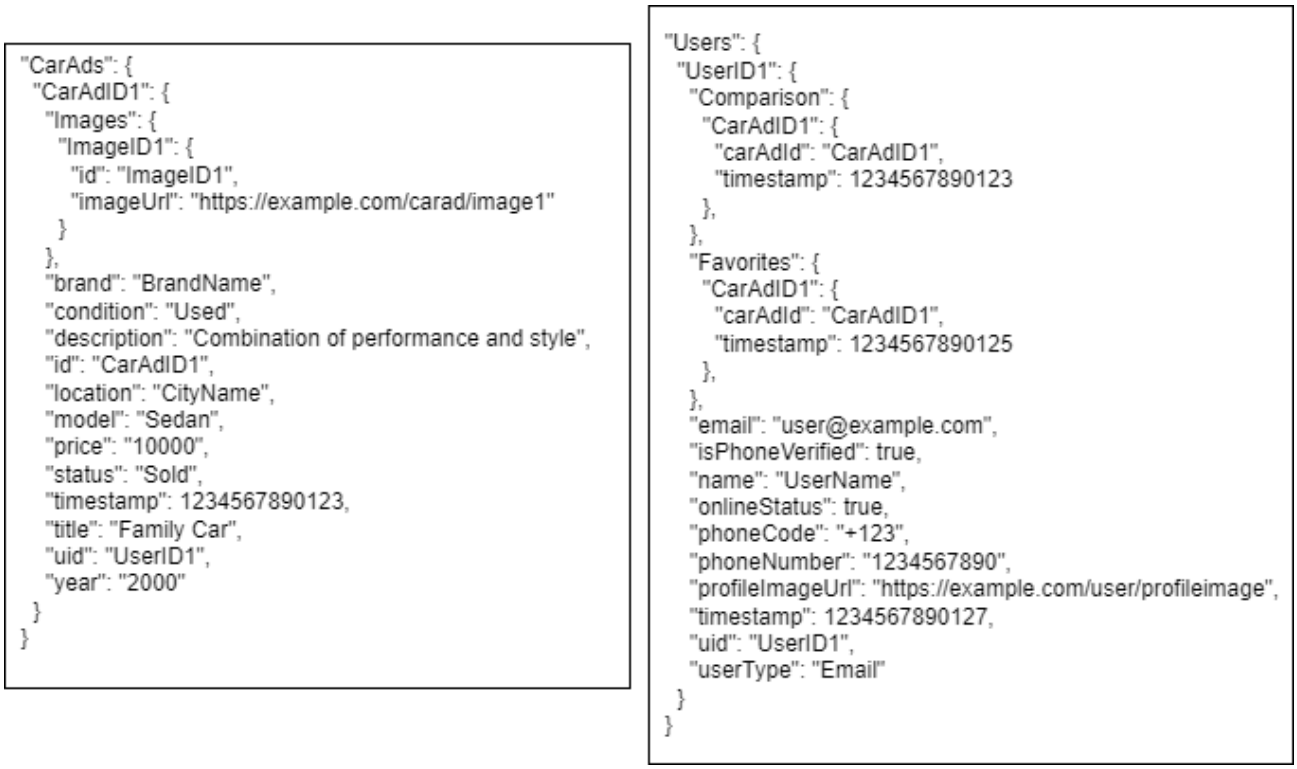


Рис. 3.5 структура бази даних у форматі JSON

Розділ Users включає наступні поля:

– унікальний ідентифікатор;

– ім'я;

– електронна пошта;

– код країни телефону;

– номер телефону;

– URL-адреса зображення профілю;

– час створення профілю у форматі UNIX;

– список улюблених автомобілів користувача, де кожен автомобіль має ідентифікатор та час додавання у вибране;

– список автомобілів для порівняння, де кожен автомобіль має ідентифікатор та час додавання у вибране.

3.6 Тестування мобільного застосунку

Тестування є дуже великою складовою мобільного застосунку, оскільки допомагає виявити помилки та неправильне функціонування, яке може вплинути негативно на застосунок.

Для тестування мобільного застосунку було обрано функціональне тестування. Функціональне тестування – це тестування, спрямоване на перевірку компонентів, щоб переконатись, що вони працюють до визначених вимог.

3.7 Перевірка функціонування мобільного застосунку

Розроблений мобільний застосунок AutoFinder потрібно перевірити на функціонування та протестувати. Усі рисунки мобільного застосунку AutoFinder, показані на емуляторі Android Studio.

3.7.1 Перевірка функціонування головної сторінки

Коли користувач запустить мобільний застосунок, перше що він зустріне буде (рис. 3.6) сторінку з логотипом програми та кнопкою «продовжити з електронною поштою», проте натискати на кнопку є не обов’язковим. Цю сторінку можна закрити та потрапити до головної сторінки.

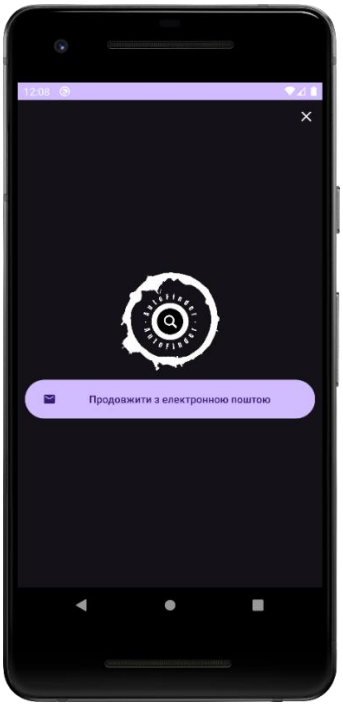


Рис. 3.6 Сторінка з логотипом

Після закриття сторінки з логотипом, користувач потрапляє на головну сторінку. На рисунку 3.7 зображено головну сторінку. На головній сторінці знаходяться такі об’єкти: поле пошуку, кнопка сортування, фільтрація за моделлю, список автомобілів. Також у низу знаходиться панель навігації між сторінками.

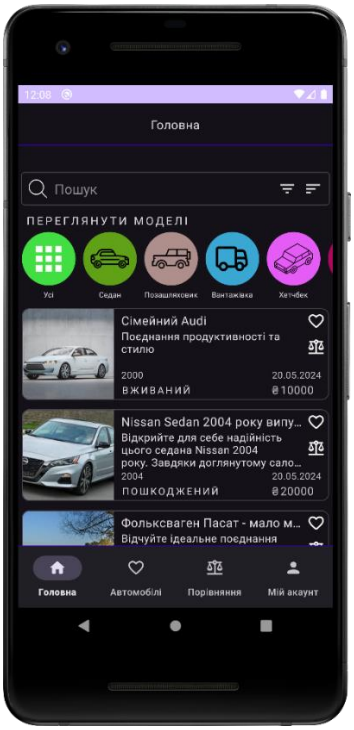


Рис. 3.7 Головна сторінка

Перевіримо функціональність фільтрації за моделлю. Натискаємо на будь-яку з моделей, які знаходяться під пошуком та над списком автомобілів. Рисунок 3.8 зображено фільтрацію моделей автомобілів типу позашляховик.

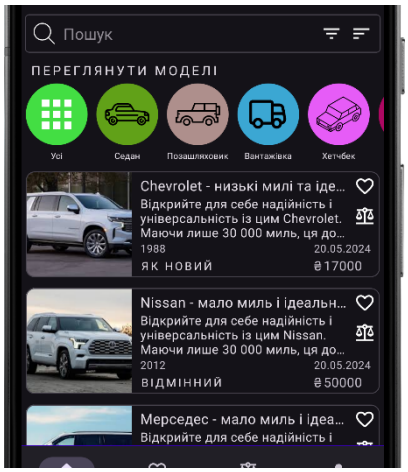


Рис. 3.8 Фільтрація моделей автомобілів типу позашляховик

Перевіримо функціональність пошуку. Натискаємо на поле вводу та вводимо бажаний текст. Результат пошуку на рисунку 3.9.

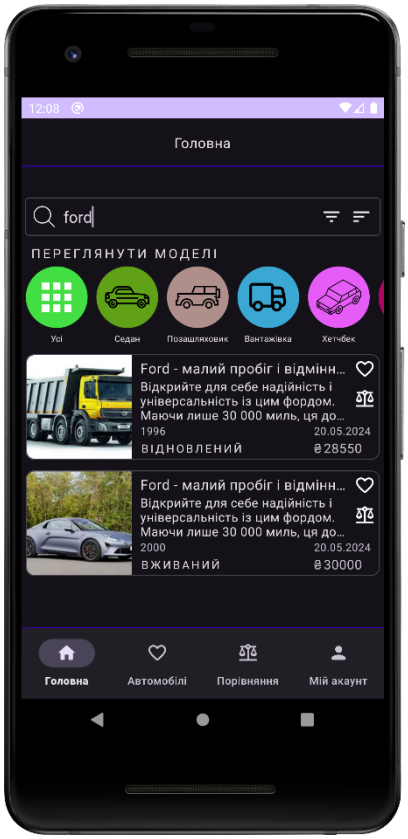


Рис. 3.9 Результат пошуку

Тепер перевіримо сортування, яке знаходиться з права у полі пошуку. Натискаємо на іконку сортування, після цього з’являється випадний список (3.10) з якого обирає сортування за ціною. Рисунок 3.11 відображає результат сортування за ціною.

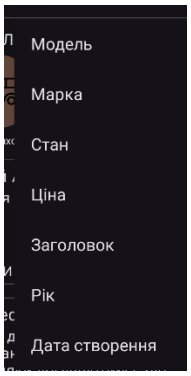


Рис. 3.10 Випадний список сортування

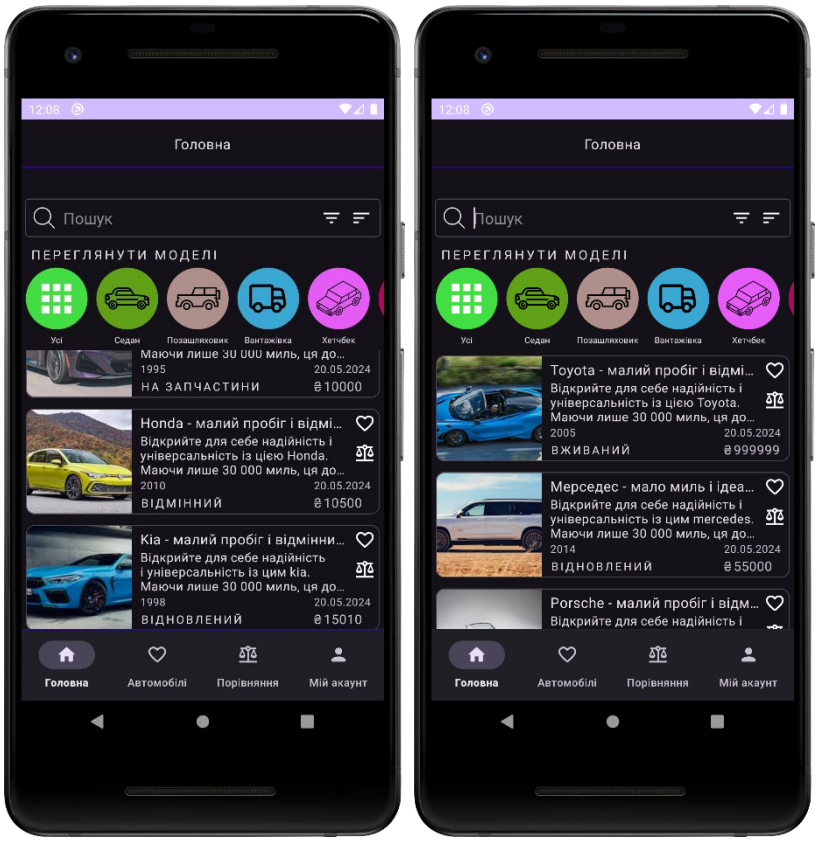


Рис. 3.11 Результат сортування за ціною

Перейдемо на детальний опис автомобіля натиснувши на автомобіль у списку. Відкривається нова сторінка (рис. 3.12)

Рис. 3.12 Сторінка детального опису автомобіля

3.7.2 Перевірка функціонування сторінки детального опису автомобіля

Перше, що знаходиться на сторінці детального опису це зображення (рис. 3.12). Кількість зображені автомобіля може бути до десяти. Щоб переглянути інші зображення потрібно горнути його у бік. Результат гортання зображення на рисунку 3.13.

Рис. 3.13 Результат гортання зображення

Під зображенням (рис. 3.12) знаходиться ціна, стан автомобіля, дата створення оголошення. Нижче можна побачити модель, марку, назву

оголошення, опис, рік випуску автомобіля, статус. Під статусом знаходиться опис користувача (рис. 3.14). Клацнемо на опис користувача. Відкривається сторінка користувача та всі його оголошення про автомобілі (рис. 3.15).

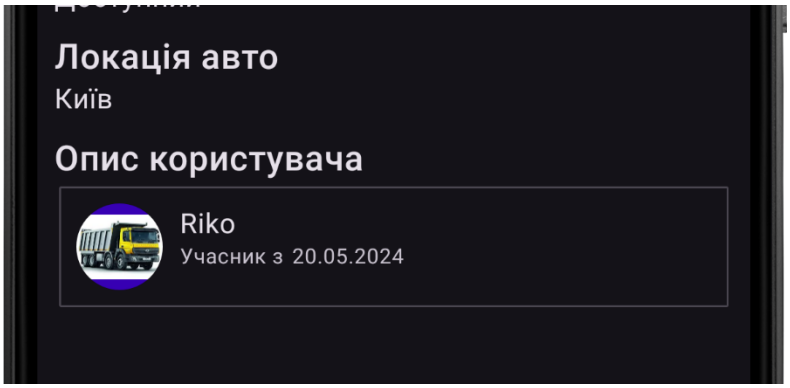


Рис. 3.14 Опис користувача

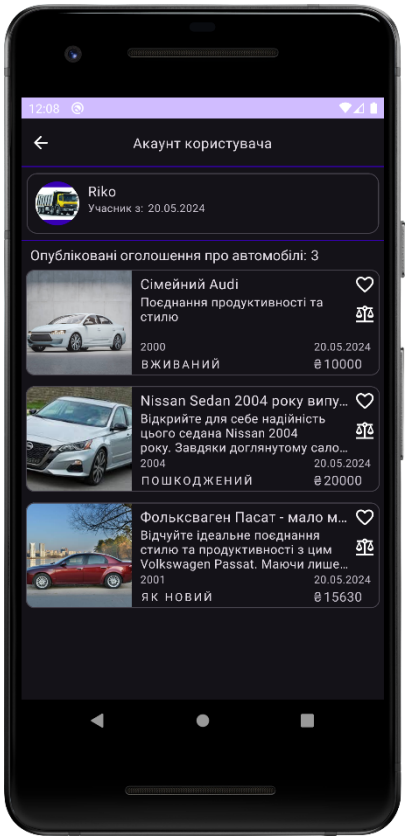


Рис. 3.15 Профіль користувача

Повернемося до сторінки з детальним описом автомобіля (рис. 3.12). Натискаємо на стрілку з направленням уліво, що знаходиться у лівому верхньому куті. У детальному описі автомобіля унизу знаходяться дві кнопки. Натиснемо на кнопку «подзвонити». Відкривається застосунок «Телефон» з введеним номером продавця, який він вказав (рис. 3.16).

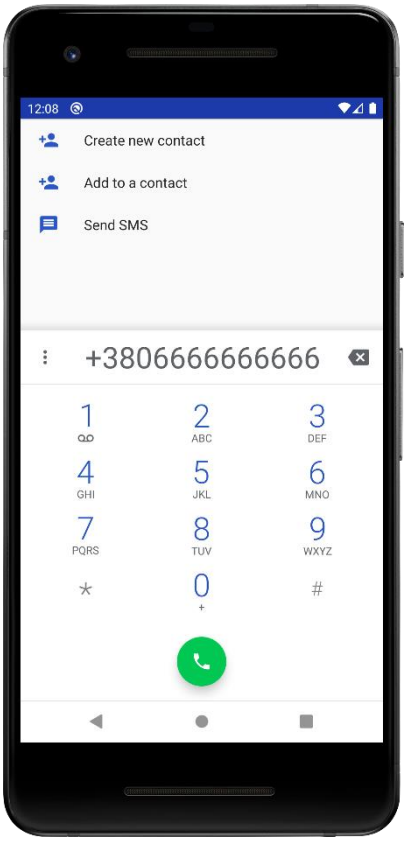


Рис. 3.16 Застосунок «Телефон» з введеним номером продавця

Повернемося та натиснемо на інше кнопку, яка називається «SMS». Відкривається мобільний застосунок «Повідомлення» у якому створений чат на номер продавця, який він вказав (рис. 3.17).

Рис. 3.17 Застосунок «Повідомлення» з чатом продавця

3.7.3 Перевірка функціонування авторизації та реєстрації

Щоб можна було додавати уподобане до списку, потрібно авторизуватися у мобільному застосунку. Повернемося на головну сторінку, та натиснемо на панелі навігації, що унизу, на іконку серця або персони. Оскільки користувач не авторизований, ці сторінки не доступні і вони переадресують користувача на сторінку з логотипом застосунку. Натиснемо на «Продовжити з електронною поштою», відкривається сторінка Увійти (рис. 3.18). Користувач, якщо він вже має профіль, може ввести дані та натиснути кнопку «увійти», якщо дані вірні користувач авторизується. Натиснемо на текст «зареєструватись». Відкриється сторінка Реєстрації (рис. 3.19), введемо потрібні дані та натиснемо кнопку, що внизу «зареєструватись».

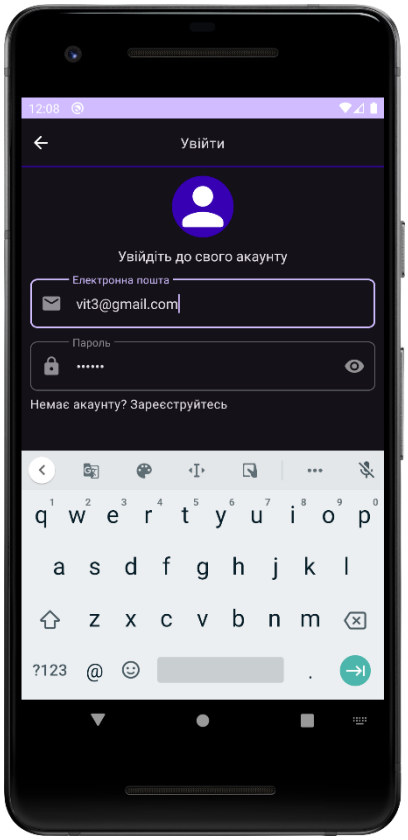


Рис. 3.18 Сторінка Увійти (з введеними даними)

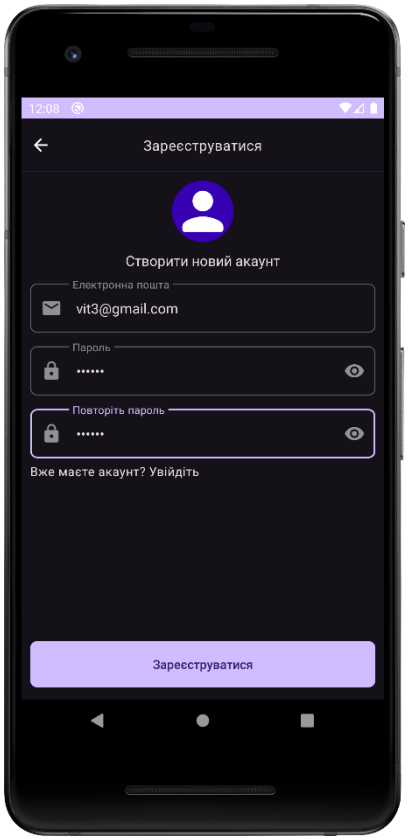


Рис. 3.19 Сторінка зареєструватись (з введеними даними)

Після натискання кнопки відбувається реєстрація профіля та автоматична авторизація.

3.7.4 Перевірка функціонування сторінки акаунту

Після створення профілю, користувач потрапляє на головну сторінку. Тепер користувач може перейти натиснувши на іконку персони, що з права у панелі навігації до перегляду власного акаунту. У акаунті містяться поля: ім’я, електронна пошта, телефон, дата народження, дата реєстрації та фото користувача.

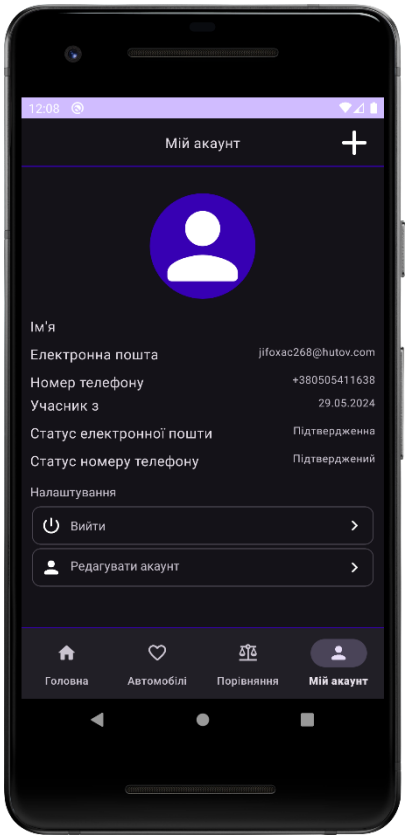


Рис. 3.20 Акаунт користувача

Натиснемо на кнопку «редагувати акаунт». Відкриється сторінка редагування акаунту (рис. 3.21).

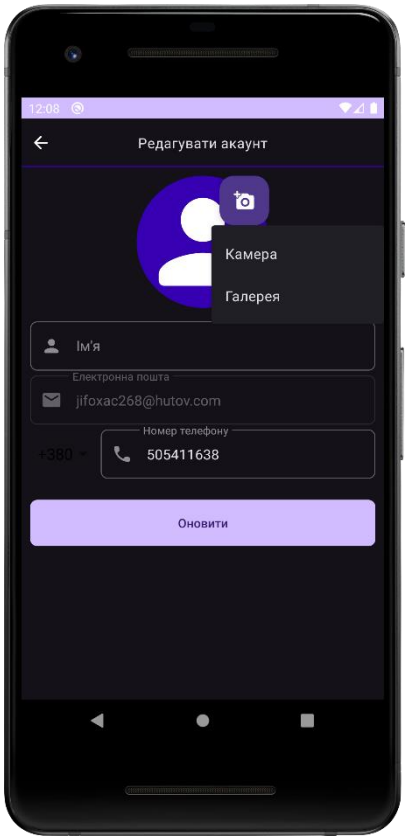


Рис. 3.21 Редагування акаунту (з введеними даними)

Натиснемо на іконку камери що розташована у лівому куті зображення користувача. З’явився випадний список (рис. 3.21). Виберемо варіант «Галерея». Вискочить повідомлення про надання доступу до фото та медіа на пристрої (рис. 3.22). Після дозволу відкриється галерея зображень, які знаходяться на пристрої (рис. 3.23).

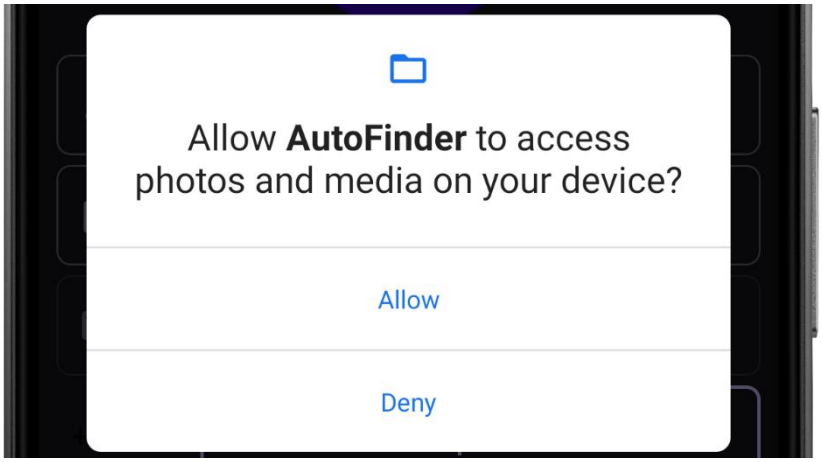


Рис. 3.22 Повідомлення про надання доступу

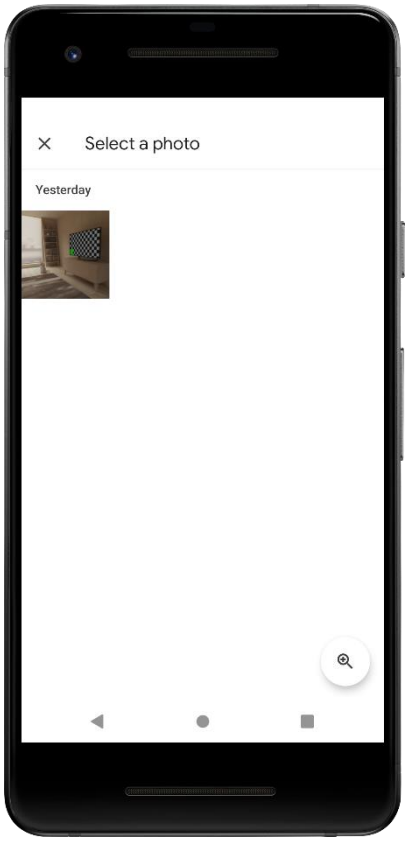


Рис. 3.23 Галерея зображень

Обираємо зображення та повертаємось назад до сторінки зміни профілю. Після натикаємо на кнопку «Оновити», дані зберігаються (рис. 3.24).

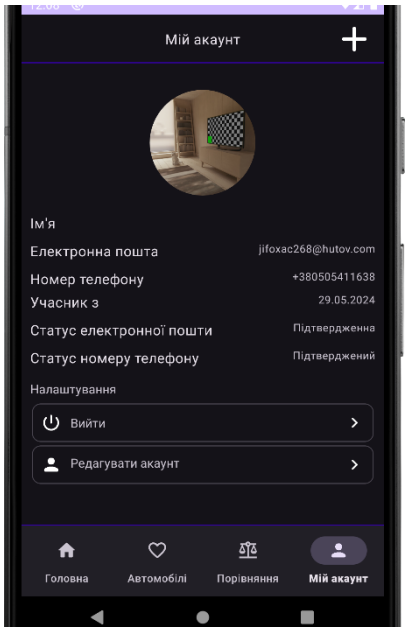


Рис. 3.24 Профіль збережених даних

3.7.5 Перевірка функціонування сторінки створення автомобільного оголошення

Щоб перевірити функціонування сторінки створення автомобільного оголошення, треба у верхньому лівому куті натиснути на іконку плюса. Це відкриє сторінку створення оголошення (рис. 3.25).

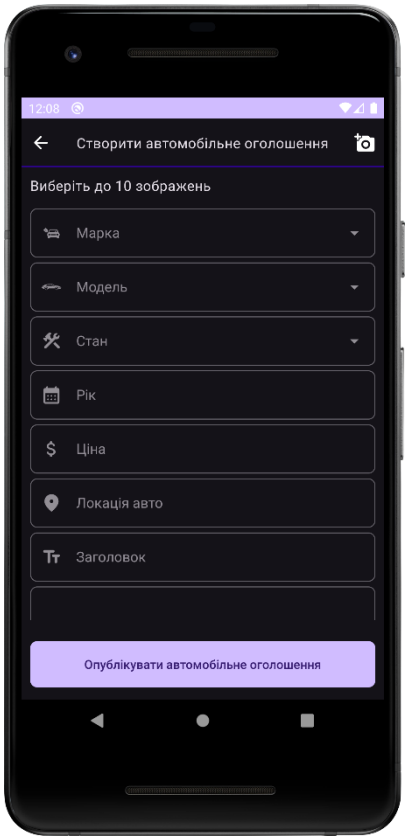


Рис. 3.25 Сторінка створення автомобільного оголошення

На сторінці створення оголошення є три випадні списки: марка, модель та стан автомобіля (рис. 3.26). Крім них є ще поля введення: рік, ціна, локація авто, назва, опис. У лівому верхньому знаходиться іконка камери, що дозволяє вибрати зображення для автомобіля з таким самим функціоналом, що на сторінці редагування профілю.

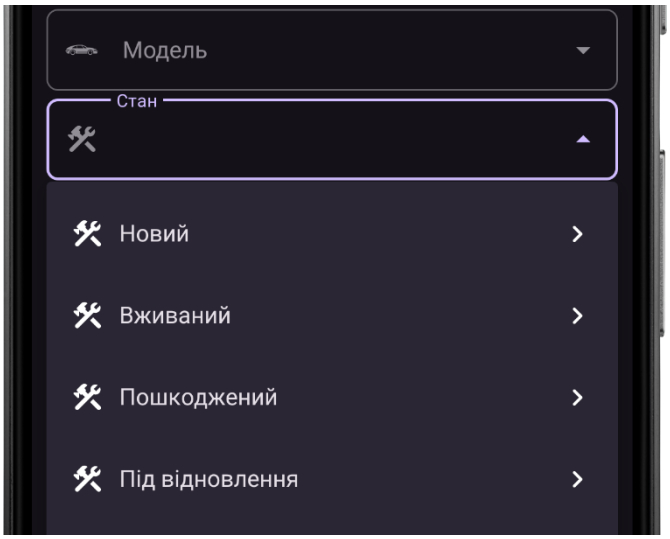


Рис. 3.26 Список стану авто

Після введення даних (рис. 3.27) натискаємо на кнопку «опублікувати оголошення автомобіля», що знаходиться внизу.

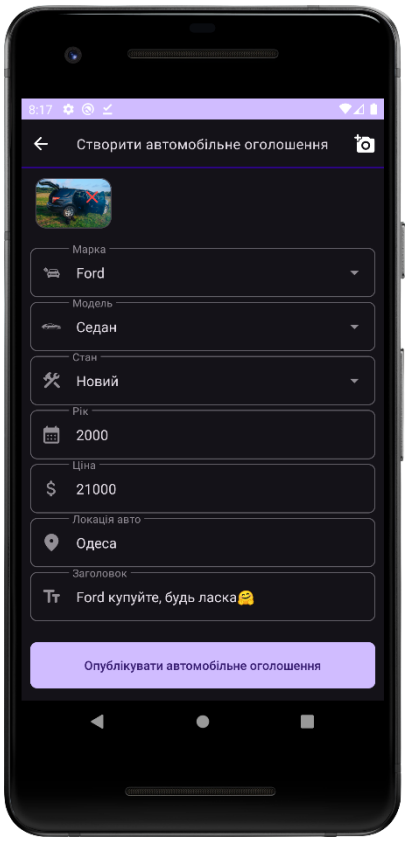


Рис. 3.27 Сторінка створення оголошення (з введеними даними)

Тепер автомобіль з’явиться на головній сторінці (рис. 3.28).

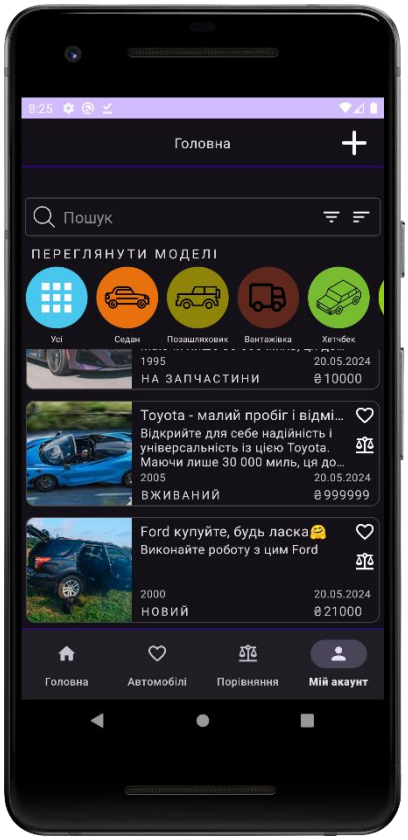


Рис. 3.28 Створене автомобільне оголошення

3.7.6 Перевірка функціонування сторінки обраних автомобілів

Спробуймо додати уподобані автомобілі до списку обраного. Натискаємо на іконку серця, яке розташоване у лівому куті автомобільного оголошення (рис. 3.29).

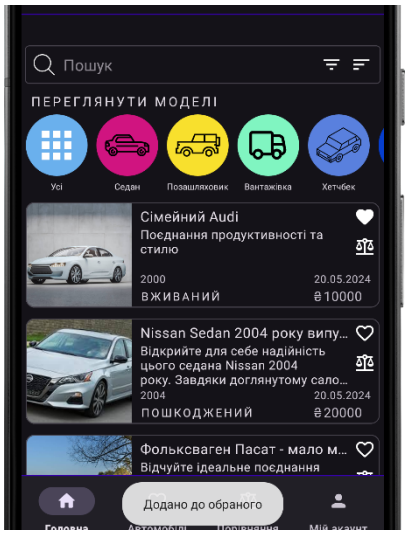


Рис. 3.29 Додавання автомобіля до списку обраного

Щоб подивиться список обраного, потрібно натиснути на іконку серця, що розташоване у панелі навігації. Це відкриє сторінку «Автомобілі». На першій вкладці розташовані автомобілі, які користувач опублікував (рис. 3.30). Перетягнемо сторінку у лівий бік. Появиться сторінка обраного з авто, що користувач додав до списку (рис. 3.31).

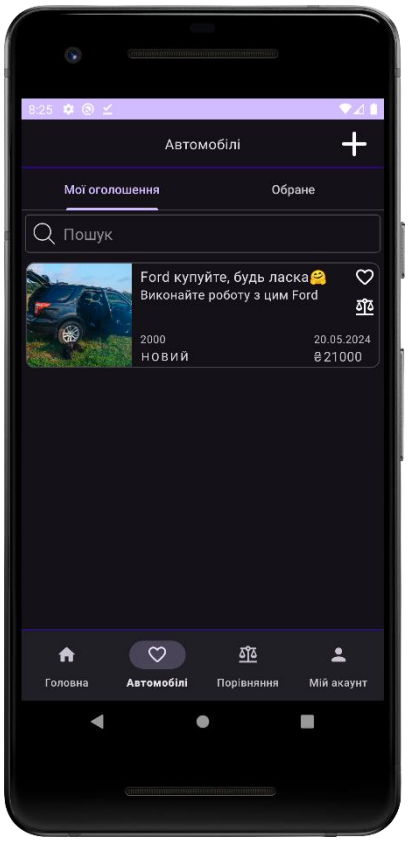


Рис. 3.30 Сторінка опублікованих автомобілів

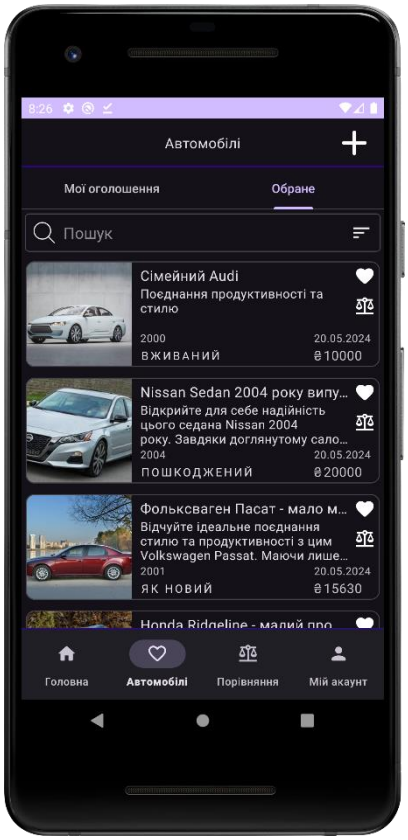


Рис. 3.31 Сторінка обраного

3.7.7 Перевірка функціонування сторінки порівняння автомобілів

Додаймо автомобілі до списку порівняння. Натискаємо на іконку терезів, яка розташована у лівому куті автомобільного оголошення під іконкою серця (рис. 3.32).

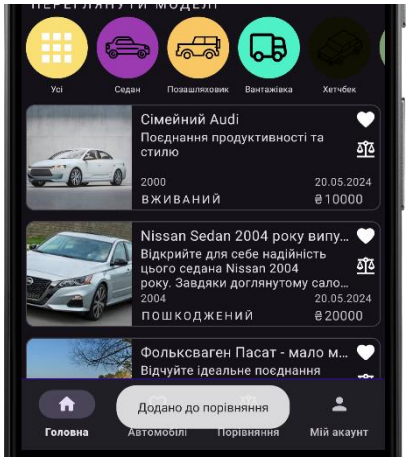


Рис. 3.32 Додавання автомобіля до списку порівняння

Щоб подивиться список порівняння, потрібно натиснути на іконку терезів, що розташовані у панелі навігації. Це відкриє сторінку «Порівняння». На сторінці розташовані автомобілі, які користувач додав до списку (рис. 3.33). Порівняння відбувається за допомогою таких полів, як: ціна, бренд, стан, модель, рік. Також зі списку порівняння автомобілі можна вилучати.

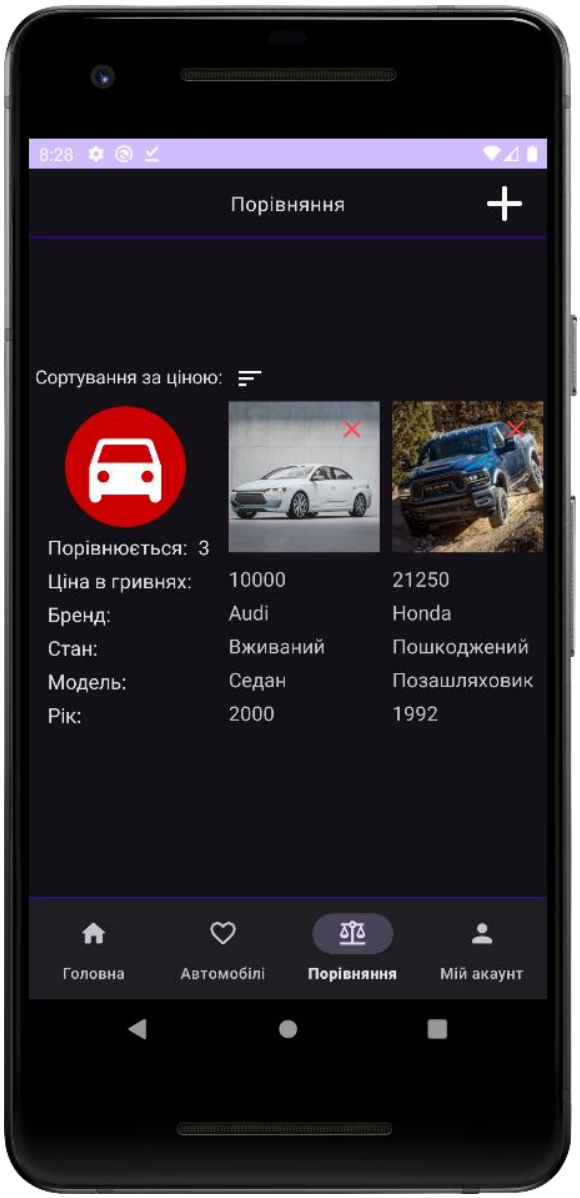


Рис. 3.33 Сторінка порівняння

ВИСНОВКИ

У кваліфікаційній роботі розроблено мобільний застосунок для пошуку та порівняння ціни автомобілів мовою Kotlin.

1. Проведено аналіз предметної галузі та проаналізовано мобільні застосунки для пошуку та порівняння ціни автомобілів. У результаті було виявлено недоліки та переваги таких застосунків, що у свою чергу сформувало вимоги до мобільного застосунку.

2. Обрано програмні засоби для реалізації мобільного застосунку: мову програмування Kotlin, Android Jetpack – набір бібліотек для спрощення процесу створення, Firebase для серверної частини.

3. Спроектовано мобільний застосунок за допомогою діаграм: прецедентів, класів, mind map. Створено схему бази даних у JSON форматі.

4. Програмно реалізовано мобільний застосунок для пошуку та порівняння ціни автомобілів мовою Kotlin.

5. Протестовано функціонал мобільного застосунку для пошуку та порівняння ціни автомобілів. Щоб виявити неправильне функціонування застосунку та запобігти можливим помилкам.

Робота пройшла апробацію:

Вдовиченко В.М., Садовенко В.С. Аналіз мобільних застосунків для пошуку та порівняння ціни автомобілів // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2024 року, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 144-145.

Вдовиченко В.М., Садовенко В.С. Розробка мобільного застосунку для пошуку та порівняння ціни автомобілів мовою Kotlin // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2024 року, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 146.

ПЕРЕЛІК ПОСИЛАНЬ

1. Android Jetpack Dev Resources. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/jetpack.
2. AUTO.RIA. [Електронний ресурс] – Режим доступу до ресурсу: https://auto.ria.com/uk/.
3. AutoScout24. [Електронний ресурс] – Режим доступу до ресурсу: https://autoscout24.com.
4. AutoUncle. [Електронний ресурс] – Режим доступу до ресурсу: https://autouncle.co.uk.
5. Firebase Documentation. [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs.
6. Kotlin Docs. [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/docs/home.html.
7. Meet Android Studio. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/intro.
8. What Is a Non-Relational Database?. [Електронний ресурс] – Режим доступу до ресурсу: https://builtin.com/data-science/non-relational-database.
9. What is git?. [Електронний ресурс] – Режим доступу до ресурсу: https://builtin.com/software-engineering-perspectives/git.
10. What Is MVVM Architecture?. [Електронний ресурс] – Режим доступу до ресурсу: https://builtin.com/software-engineering-perspectives/mvvm-architecture.
11. What Is XML?. [Електронний ресурс] – Режим доступу до ресурсу: https://builtin.com/data-science/xml.
12. Хортон Д. Розробка Android-додатків з нуля / Джон Хортон. – Снятині: Print2print, 2023. – 576 с. – (3).

13. Laurencece P. Programming Android with Kotlin. Achieving Structured Concurrency with Coroutines. / P. Laurencece, H. Amanda. – Sebastopol: O\'Reilly, 2021. – 336 с. – (1).

14. Fazio M. Kotlin and Android Development featuring Jetpack. Build Better, Safer Android Apps / Michael Fazio. – New York: BHV-СПб, 2021. – 446 с.

15. Kousen K. Gradle Recipes for Android: Master the New Build System for Android / Ken Kousen. – Sebastopol: O\'Reilly, 2016. – 168 с. – (1).

16. Wilson J. Creating Dynamic UI with Android Fragments / Jim R. Wilson. – Birmingham: Packt Publishing, 2013. – 122 с.

17. Moroney L. The Definitive Guide to Firebase: Build Android Apps on Google\'s Mobile Platform / Laurence Moroney. – New York: Apress, 2017. – 288 с. – (1).

18. Soshin A. otlin Design Patterns and Best Practices - Third Edition: Elevate your Kotlin skills with classical and modern design patterns, coroutines, and microservices / Alexey Soshin. – Birmingham: Packt Publishing, 2024. – 474 с. – (3).

19. Kouraklis J. MVVM in Delphi: Architecting and Building Model View ViewModel Applications / John Kouraklis. – New York: Apress, 2016. – 308 с.

20. Snider E. Mastering Xamarin.Forms: Build Rich, Maintainable, and Cross-Platform Mobile Apps with Xamarin.Forms / E. Snider. – Packt Publishing: Birmingham, 2018. – 244 с.

21. Straub B. Pro Git / B. Straub, S. Chacon. – New York: Apress, 2014. – 456 с.

22. Sadalaj P. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence / Pramod Sadalaj. – Boston: Addison-Wesley, 2012. – 192 с.

23. Skeen J. Kotlin Programming: The Big Nerd Ranch Guide / J. Skeen, D. Greenhalgh. – Atlanta: Big Nerd Ranch Guides, 2018. – 480 с.

24. Jemerov D. Kotlin in Action / D. Jemerov, S. Isakova. – Shelter Island: Manning Publications, 2017. – 360 с.

25. Urbanowicz S. Android Development with Kotlin: Quick Start Guide / Samuel Urbanowicz. – Birmingham: Packt Publishing, 2018. – 194 с.

26. Smyth N. Firebase Essentials - Android Edition / Neil Smyth. – Rochester: Payload Media, 2021. – 348 с.

27. Subramaniam V. Programming Kotlin / Venkat Subramaniam. – Dallas: Pragmatic Bookshelf, 2019. – 272 с.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка мобільного застосунку для пошуку та порівняння ціни автомобілів мовою Kotlin

Виконав студент 4 курсу
групи ПД-44
Вдовиченко Віталій Миколайович
Керівник роботи
к.ф.-м.н., доцент,
професор кафедри ІПЗ Садовенко Володимир Сергійович

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – спрощення процесу пошуку та порівняння автомобілів за рахунок застосування мобільного додатку, створеного на мові Kotlin.

Об'єкт дослідження – процес пошуку та порівняння ціни автомобілів.

Предмет дослідження – програмне забезпечення для пошуку та порівняння ціни автомобілів.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Аналіз існуючих аналогів пошуку та порівняння ціни автомобілів.
2. Формування вимог до мобільного застосунку для пошуку та порівняння ціни автомобілів.
3. Проектування мобільного застосунку для пошуку та порівняння ціни автомобілів.
4. Програмна реалізація мобільного застосунку для пошуку та порівняння ціни автомобілів.
5. Тестування мобільного застосунку.

АНАЛІЗ АНАЛОГІВ

Функціонал	AUTO.RIA	AutoUncle	AutoScout24	AutoFinder
Розширений пошук	+	+	+	+
Порівняння автомобілів	-	-	-	+
Розміщення оголошення про продаж авто	+	-	+	+
Додавання до списку вподобаного	+	+	+	+
Створення власного аккаунту	+	+	+	+
Редагування профілю	+	-	+	+

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні:

1) пошук автомобілів;
2) додавання автомобілів до порівняння;
3) додавання автомобілів до списку бажаного;
4) створення власного профілю;
5) фільтрування оголошень про продаж автомобілів;
6) перегляд оголошення детально;
7) перегляд профілю користувача;
8) створення власного оголошення про продаж автомобілів

Нефункціональні:

1) платформа Android;
2) мова українська;
3) за датою створення оголошення сортування за замовчуванням.

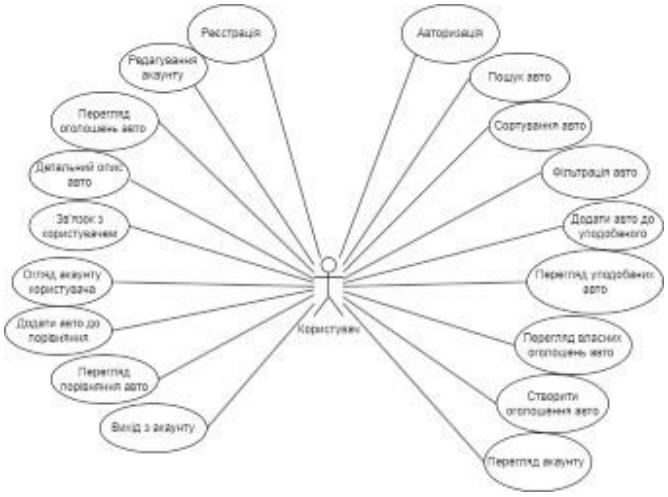
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Kotlin

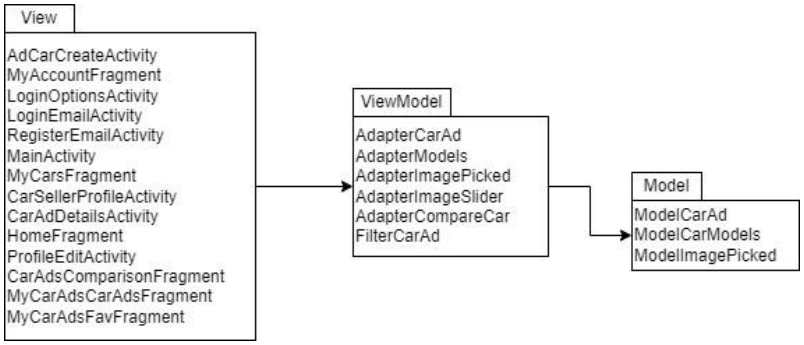
Android Studio

Android Jetpack

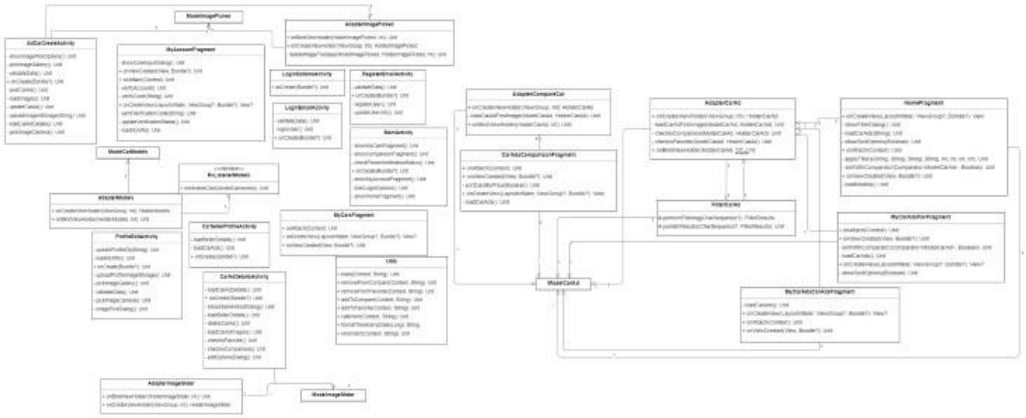
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



АРХІТЕКТУРА ЗАСТОСУНКУ



ДІАГРАМА КЛАСІВ



СТРУКТУРА JSON ФАЙЛУ


```
"CarAds": {
  "CarAdID1": {
    "Images": {
      "ImageID1": {
        "id": "ImageID1",
        "imageUrl": "https://example.com/carad/image1"
      }
    },
    "brand": "BrandName",
    "condition": "Used",
    "description": "Combination of performance and style",
    "id": "CarAdID1",
    "location": "CityName",
    "model": "Sedan",
    "price": "10000",
    "status": "Sold",
    "timestamp": 1234567890123,
    "title": "Family Car",
    "uid": "UserID1",
    "year": "2000"
  }
}
```

```
"Users": {
  "UserID1": {
    "Comparison": {
      "CarAdID1": {
        "carAdId": "CarAdID1",
        "timestamp": 1234567890123
      },
    },
    "Favorites": {
      "CarAdID1": {
        "carAdId": "CarAdID1",
        "timestamp": 1234567890125
      },
    },
    "email": "user@example.com",
    "isPhoneVerified": true,
    "name": "UserName",
    "onlineStatus": true,
    "phoneCode": "+123",
    "phoneNumber": "1234567890",
    "profileImageUrl": "https://example.com/user/profileimage",
    "timestamp": 1234567890127,
    "uid": "UserID1",
    "userType": "Email"
  }
}
```

ЕКРАННІ ФОРМИ

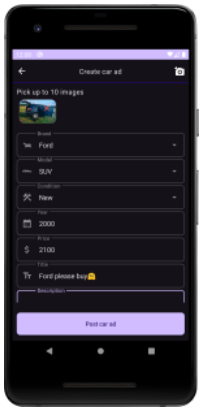
Головна сторінка

Розширений пошук

Порівняння автомобілів

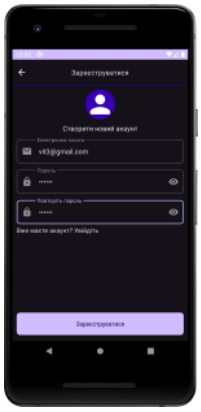
Детальний опис авто

ЕКРАННІ ФОРМИ

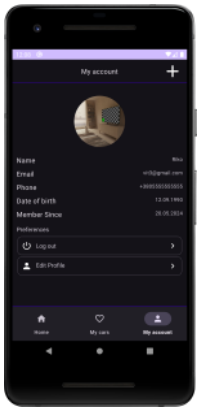


Створення автомобільного оголошення

Список обраного

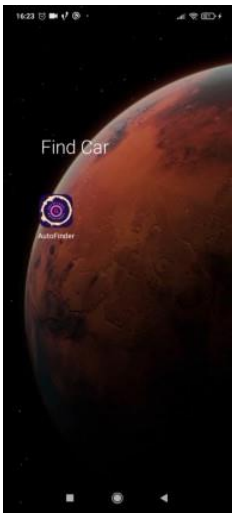


Реєстрація



Профіль користувача

ДЕМОНСТРАЦІЯ ЗАСТОСУНКУ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Вдовиченко В.М., Садовенко В.С. Аналіз мобільних застосунків для пошуку та порівняння ціни автомобілів // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник. тез 24 квітня 2024 року, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 144-145.

2. Вдовиченко В.М., Садовенко В.С. Розробка мобільного застосунку для пошуку та порівняння ціни автомобілів мовою Kotlin // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник. тез 24 квітня 2024 року, ДУІКТ, м. Київ. К: ДУІКТ, 2024. С. 146.

ВИСНОВКИ

1. Проведено аналіз предметної галузі та проаналізовано мобільні застосунки для пошуку та порівняння ціни автомобілів. У результаті було виявлено недоліки та переваги таких застосунків, що у свою чергу сформувало вимоги до мобільного застосунку.
2. Спроектовано мобільний застосунок за допомогою шаблону проектування MVVM, діаграми варіантів використання та класів.
3. Програмно реалізовано мобільний застосунок для пошуку та порівняння ціни автомобілів за допомогою таких інструментів: мова програмування Kotlin, Android Jetpack – набір бібліотек для спрощення процесу створення, Firebase для серверної частини.
4. Протестовано функціонал компонентів мобільного застосунку для пошуку та порівняння ціни автомобілів. Щоб виявити неправильне функціонування застосунку та запобігти можливим помилкам.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
package com.example.auto_finder.activities

import android.Manifest
import android.app.Activity
import android.app.ProgressDialog
import android.content.ContentValues
import android.content.Intent
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.provider.MediaStore
import android.util.Log
import android.view.Menu
import android.widget.ArrayAdapter
import android.widget.PopupMenu
import
androidx.activity.result.contract.ActivityResultContract
s
import androidx.appcompat.app.AppCompatActivity
import
com.example.auto_finder.adapters.AdapterImagePicke
d
import
com.example.auto_finder.models.ModelImagePicked
import com.example.auto_finder.R
import com.example.auto_finder.Utils
import
com.example.auto_finder.databinding.ActivityAdCarC
reateBinding
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.FirebaseDatabase
import
com.google.firebase.database.ValueEventListener
import com.google.firebase.storage.FirebaseStorage
import kotlin.math.log


class AdCarCreateActivity : AppCompatActivity() {

    private lateinit var binding:
ActivityAdCarCreateBinding

    private companion object {
        private const val TAG =
"AD_CAR_CREATE_TAG"
    }

    private lateinit var progressDialog: ProgressDialog

    private lateinit var firebaseAuth: FirebaseAuth

    private var imageUri: Uri? = null

    private lateinit var imagePickedArrayList:
ArrayList<ModelImagePicked>

    private lateinit var adapterImagePicked:
AdapterImagePicked

    private var isEditMode = false
    private var carAdIdForEditing = ""

    override fun onCreate(savedInstanceState: Bundle?)
{
        super.onCreate(savedInstanceState)

        binding =
ActivityAdCarCreateBinding.inflate(layoutInflater)
        setContentView(binding.root)

        progressDialog = ProgressDialog(this)
        progressDialog.setTitle("Зачекайте")

progressDialog.setCanceledOnTouchOutside(false)

        firebaseAuth = FirebaseAuth.getInstance()



        val adapterBrands = ArrayAdapter(this,
R.layout.row_brand_act, Utils.brands)
        binding.brandAct.setAdapter(adapterBrands)

        val adapterModels = ArrayAdapter(this,
R.layout.row_model_act, Utils.models)
        binding.modelAct.setAdapter(adapterModels)

        val adapterConditions = ArrayAdapter(this,
R.layout.row_condition_act, Utils.conditions)

binding.conditionAct.setAdapter(adapterConditions)

        isEditMode =
intent.getBooleanExtra("isEditMode", false)
        Log.d(TAG, "onCreate: isEditMode:
$isEditMode")

        if (isEditMode){
            carAdIdForEditing =
intent.getStringExtra("carAdId") ?: ""

            loadCarAdDetails()

            binding.toolbarTitleTv.text = "Оновити
автомобільне оголошення"
```

```
            binding.postCarAdBtn.text = "Оновити автомобільне оголошення"
        }else{
            binding.toolbarTitleTv.text = "Створити автомобільне оголошення"
            binding.postCarAdBtn.text = "Опублікувати автомобільне оголошення"
        }

        imagePickedArrayList = ArrayList()

        loadImages()

        binding.toolbarBackBtn.setOnClickListener {
            onBackPressed()
        }

        binding.toolbarAddImageBtn.setOnClickListener {
            showImagePickOptions()
        }

        binding.postCarAdBtn.setOnClickListener{
            validateData()
        }
    }

    private fun loadImages(){
        Log.d(TAG, "loadImages: ")

        adapterImagePicked = AdapterImagePicked(this, imagePickedArrayList, carAdIdForEditing)

        binding.imagesRv.adapter = adapterImagePicked
    }

    private fun showImagePickOptions(){
        Log.d(TAG, "showImagePickOptions: ")
        val popupMenu = PopupMenu(this, binding.toolbarAddImageBtn)

        popupMenu.menu.add(Menu.NONE, 1, 1, "Камера")
        popupMenu.menu.add(Menu.NONE, 2, 2, "Галерея")

        popupMenu.show()

        popupMenu.setOnMenuItemClickListener { item ->
            val itemId = item.itemId

            if (itemId == 1){
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU){
                    val cameraPermissions = arrayOf(Manifest.permission.CAMERA)
                    requestCameraPermission.launch(cameraPermissions)
                }else{
                    val cameraPermission = arrayOf(Manifest.permission.CAMERA, Manifest.permission.WRITE_EXTERNAL_STORAGE)

                    requestCameraPermission.launch(cameraPermission)
                }
            }else if(itemId == 2){
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU){
                    pickImageGallery()
                }else{
                    val storagePermission = Manifest.permission.WRITE_EXTERNAL_STORAGE

                    requestStoragePermission.launch(storagePermission)
                }
            }
            true
        }
    }

    private val requestStoragePermission = registerForActivityResult(
        ActivityResultContracts.RequestPermission()
    ){isGranted ->
        Log.d(TAG, "requestStoragePermission: isGranted $isGranted")

        if (isGranted){
            pickImageGallery()
        }else{
            Utils.toast(this, "Відмовлено у дозволі на доступ до зберігання")
        }
    }

    private val requestCameraPermission = registerForActivityResult(

        ActivityResultContracts.RequestMultiplePermissions()
    ){ result ->
        Log.d(TAG, "requestCameraPermission: result: $result")

        var areAllGranted = true
        for (isGranted in result.values){
            areAllGranted = areAllGranted && isGranted
        }

        if (areAllGranted){
            pickImageCamera()
        } else {
            Utils.toast(this, "Дозвіл на доступ до камери або сховище відхилено або обидва")
```

```
        }

    }

    private fun pickImageGallery(){
        Log.d(TAG, "pickImageGallery: ")

        val intent = Intent(Intent.ACTION_PICK)
        intent.type = "image/*"
        galleryActivityResultLauncher.launch(intent)
    }

    private fun pickImageCamera(){
        Log.d(TAG, "pickImageCamera: ")

        val contentValues = ContentValues()

contentValues.put(MediaStore.Images.Media.TITLE,
"TEMP_IMAGE_TITLE")

contentValues.put(MediaStore.Images.Media.DESCRI
PTION, "TEMP_IMAGE_DESCRIPTION")

        imageUri =
contentResolver.insert(MediaStore.Images.Media.EXT
ERNAL_CONTENT_URI, contentValues)

        val intent =
Intent(MediaStore.ACTION_IMAGE_CAPTURE)
        intent.putExtra(MediaStore.EXTRA_OUTPUT,
imageUri)
        cameraActivityResultLauncher.launch(intent)
    }

    private val galleryActivityResultLauncher =
registerForActivityResult(
        ActivityResultContracts.StartActivityForResult()
    ){ result ->
        Log.d(TAG, "galleryActivityResultLauncher: ")

        if (result.resultCode == Activity.RESULT_OK){
            val data = result.data
            imageUri = data!!.data
            Log.d(TAG, "galleryActivityResultLauncher:
imageUri: $imageUri")

            val timestamp = "${Utils.getTimestamp()}"
            val modelImagePicked =
ModelImagePicked(timestamp, imageUri, null, false)

            imagePickedArrayList.add(modelImagePicked)

            loadImages()
        }else{
            Utils.toast(this, "Відхилено!")
        }
    }

    private val cameraActivityResultLauncher =
registerForActivityResult(
        ActivityResultContracts.StartActivityForResult()
    ){ result ->
        Log.d(TAG, "cameraActivityResultLauncher: ")
        if (result.resultCode == Activity.RESULT_OK){
            Log.d(TAG, "cameraActivityResultLauncher:
imageUri $imageUri")

            val timestamp = "${Utils.getTimestamp()}"
            val modelImagePicked =
ModelImagePicked(timestamp, imageUri, null, false)

            imagePickedArrayList.add(modelImagePicked)

            loadImages()
        } else{
            Utils.toast(this, "Відхилено!")
        }
    }

    private var brand = ""
    private var model = ""
    private var condition = ""
    private var year = ""
    private var price = ""
    private var location = ""
    private var title = ""
    private var description = ""

    private fun validateData() {
        Log.d(TAG, "validateData: ")

        brand = binding.brandAct.text.toString().trim()
        model = binding.modelAct.text.toString().trim()
        condition =
binding.conditionAct.text.toString().trim()
        year = binding.yearEt.text.toString().trim()
        price = binding.priceEt.text.toString().trim()
        location = binding.locationEt.text.toString().trim()
        title = binding.titleEt.text.toString().trim()
        description =
binding.descriptionEt.text.toString().trim()

        if (brand.isEmpty()) {
            binding.brandAct.error = "Оберіть марку"
            binding.brandAct.requestFocus()
        } else if (model.isEmpty()) {
            binding.modelAct.error = "Оберіть модель"
            binding.modelAct.requestFocus()
        } else if (condition.isEmpty()) {
            binding.conditionAct.error = "Оберіть стан"
            binding.conditionAct.requestFocus()
        } else if (year.isEmpty()) {
            binding.yearEt.error = "Введіть рік"
            binding.yearEt.requestFocus()
        } else if (price.isEmpty()) {
            binding.priceEt.error = "Введіть ціну"
```

```
            binding.priceEt.requestFocus()
        }else if (location.isEmpty()) {
            binding.locationEt.error = "Введіть локацію
авто"
            binding.locationEt.requestFocus()
        } else if (title.isEmpty()) {
            binding.titleEt.error = "Введіть заголовок"
            binding.titleEt.requestFocus()
        } else if (description.isEmpty()) {
            binding.descriptionEt.error = "Введіть опис"
            binding.descriptionEt.requestFocus()
        } else {
            if (isEditMode){
                updateCarAd()
            }else{
                postCarAd()
            }
        }
    }

    private fun postCarAd(){
        Log.d(TAG, "postCarAd: ")

        progressDialog.setMessage("Публікація
автомобільного оголошення")

        progressDialog.show()

        val timestamp = Utils.getTimestamp()
        val refCarAds =
FirebaseDatabase.getInstance().getReference("CarAds"
)
        val keyId = refCarAds.push().key

        val hashMap = HashMap<String, Any>()
        hashMap["id"] = "$keyId"
        hashMap["uid"] = "${firebaseAuth.uid}"
        hashMap["brand"] = "$brand"
        hashMap["model"] = "$model"
        hashMap["condition"] = "$condition"
        hashMap["year"] = "$year"
        hashMap["price"] = "$price"
        hashMap["location"] = "$location"
        hashMap["title"] = "$title"
        hashMap["description"] = "$description"
        hashMap["status"] =
"${Utils.CAR_AD_STATUS_AVAILABLE}"
        hashMap["timestamp"] = timestamp

        refCarAds.child(keyId!!)
            .setValue(hashMap)
            .addOnSuccessListener {
                Log.d(TAG, "postCarAd: Car Ad
Published")
                uploadImagesStorage(keyId)
            }
            .addOnFailureListener {e ->
                Log.e(TAG, "postCarAd: ", e)
                progressDialog.dismiss()
                Utils.toast(this, " Не вдалося через
${e.message}")
            }
    }

    private fun updateCarAd(){
        Log.d(TAG, "updateCarAd: ")

        progressDialog.setMessage("Оновлення
оголошення...")
        progressDialog.show()

        val hashMap = HashMap<String, Any>()
        hashMap["brand"] = "$brand"
        hashMap["model"] = "$model"
        hashMap["condition"] = "$condition"
        hashMap["year"] = "$year"
        hashMap["price"] = "$price"
        hashMap["location"] = "$location"
        hashMap["title"] = "$title"
        hashMap["description"] = "$description"

        val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
        ref.child(carAdIdForEditing)
            .updateChildren(hashMap)
            .addOnSuccessListener {
                Log.d(TAG, "updateCarAd: Car ad
updated...")
                progressDialog.dismiss()
                uploadImagesStorage(carAdIdForEditing)
            }
            .addOnFailureListener { e ->
                Log.e(TAG, "updateCarAd: ", e)
                progressDialog.dismiss()
                Utils.toast(this, "Не вдалося оновити
автомобільне оголошення через ${e.message}")
            }
    }

    private fun uploadImagesStorage(adId: String){
        for (i in imagePickedArrayList.indices){
            val modelImagePicked =
imagePickedArrayList[i]

            if (!modelImagePicked.fromInternet){
                val imageName = modelImagePicked.id
                val filePathAndName =
"CarAds/$imageName"
                val imageIndexForProgress = i + 1

                val storageReference =
FirebaseStorage.getInstance().getReference(filePathAn
dName)

storageReference.putFile(modelImagePicked.imageUri
```

```
!!)
            .addOnProgressListener {snapshot ->
                val progress = 100.0 *
snapshot.bytesTransferred / snapshot.totalByteCount
                Log.d(TAG, "uploadImagesStorage:
progress: $progress")
                val message = "Завантаження
$imageIndexForProgress of
${imagePickedArrayList.size} зображень... Прогрес
${progress.toInt()}"
                Log.d(TAG, "uploadImagesStorage:
message: $message")

                progressDialog.setMessage(message)
                progressDialog.show()

            }
            .addOnSuccessListener {taskSnapshot ->
                Log.d(TAG, "uploadImagesStorage:
onSuccess")

                val uriTask =
taskSnapshot.storage.downloadUrl
                while (!uriTask.isSuccessful);
                val uploadedImageUrl = uriTask.result

                if (uriTask.isSuccessful){
                    val hashMap = HashMap<String,
Any>()
                    hashMap["id"] =
"${modelImagePicked.id}"
                    hashMap["imageUrl"] =
"$uploadedImageUrl"

                    val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
                    ref.child(adId).child("Images")
                        .child(imageName)
                        .updateChildren(hashMap)
                    }
                progressDialog.dismiss()
            }
            .addOnFailureListener{e ->
                Log.e(TAG, "uploadImagesStorage: ",
e)
                progressDialog.dismiss()
            }
        }

    }
  }

  private fun loadCarAdDetails(){
    Log.d(TAG, "loadCarAdDetails: ")

    val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
    ref.child(carAdIdForEditing)
        .addListenerForSingleValueEvent(object
:ValueEventListener{
            override fun onDataChange(snapshot:
DataSnapshot) {
                val brand =
"${snapshot.child("brand").value}"
                val model =
"${snapshot.child("model").value}"
                val condition =
"${snapshot.child("condition").value}"
                val year =
"${snapshot.child("year").value}"
                val price =
"${snapshot.child("price").value}"
                val location =
"${snapshot.child("location").value}"
                val title =
"${snapshot.child("title").value}"
                val description =
"${snapshot.child("description").value}"

                binding.brandAct.setText(brand)
                binding.modelAct.setText(model)
                binding.conditionAct.setText(condition)
                binding.yearEt.setText(year)
                binding.priceEt.setText(price)
                binding.locationEt.setText(location)
                binding.titleEt.setText(title)
                binding.descriptionEt.setText(description)

                val refImages =
snapshot.child("Images").ref

refImages.addListenerForSingleValueEvent(object
:ValueEventListener{
                    override fun onDataChange(snapshot:
DataSnapshot) {
                        for (ds in snapshot.children){
                            val id = "${ds.child("id").value}"
                            val imageUrl =
"${ds.child("imageUrl").value}"

                            val modelImagePicked =
ModelImagePicked(id, null, imageUrl, true)

imagePickedArrayList.add(modelImagePicked)
                        }
                        loadImages()
                    }

                    override fun onCancelled(error:
DatabaseError) {

                    }
                })
            }
```

```
            override fun onCancelled(error:
DatabaseError) {

            }
        })
    }
}

package com.example.auto_finder.activities

import android.content.Intent
import android.util.Log
import
com.google.android.material.dialog.MaterialAlertDialo
gBuilder
import android.os.Bundle
import android.view.Menu
import androidx.appcompat.app.AppCompatActivity
import android.view.View
import android.widget.PopupMenu
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.auth.FirebaseAuth
import com.bumptech.glide.Glide
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.DataSnapshot
import com.example.auto_finder.Utils
import
com.google.firebase.database.ValueEventListener
import com.example.auto_finder.R
import
com.example.auto_finder.adapters.AdapterImageSlider
import
com.example.auto_finder.models.ModelImageSlider
import com.example.auto_finder.models.ModelCarAd
import
com.example.auto_finder.databinding.ActivityCarAdD
etailsBinding

class CarAdDetailsActivity : AppCompatActivity() {

    private lateinit var binding:
ActivityCarAdDetailsBinding

    private companion object{
        private const val TAG =
"CAR_AD_DETAILS_TAG"
    }

    private lateinit var firebaseAuth: FirebaseAuth

    private var carAdId = ""
    private var sellerUid = ""
    private var sellerPhone = ""
    private var favorite = false
    private var comparison = false

    private lateinit var imageStringArrayList:
ArrayList<ModelImageSlider>

    override fun onCreate(savedInstanceState: Bundle?)
{
        super.onCreate(savedInstanceState)
        binding =
ActivityCarAdDetailsBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.toolbarEditBtn.visibility = View.GONE
        binding.toolbarDeleteBtn.visibility = View.GONE
        binding.callBtn.visibility = View.GONE
        binding.smsBtn.visibility = View.GONE

        firebaseAuth = FirebaseAuth.getInstance()

        carAdId =
intent.getStringExtra("carAdId").toString()
        Log.d(TAG, "onCreate: carAdId: $carAdId")

        if (firebaseAuth.currentUser != null){
            checkIsFavorite()
            checkIsComparison()
        }

        loadCarAdDetails()
        loadCarAdImages()

        binding.toolbarBackBtn.setOnClickListener {
            onBackPressed()
        }

        binding.toolbarDeleteBtn.setOnClickListener {
            val materialAlertDialogBuilder =
MaterialAlertDialogBuilder(this)
            materialAlertDialogBuilder.setTitle("Видалити
оголошення про автомобіль")
                .setMessage("Ви впевнені, що хочете
видалити цей автомобіль?")
                .setPositiveButton("Видалити"){dialog,
which ->
                    Log.d(TAG, "onCreate: Натиснено
видалити...")
                    deleteCarAd()
                }
                .setNegativeButton("Скасувати"){dialog,
which ->
                    Log.d(TAG, "onCreate: Натиснено
скасувати....")
                    dialog.dismiss()
                }
                .show()
        }

        binding.toolbarEditBtn.setOnClickListener {
            editOptionsDialog()
        }

        binding.toolbarFavBtn.setOnClickListener {
```

```
            if (favorite){
                Utils.removeFromFavorite(this, carAdId)
            }else{
                Utils.addToFavorite(this, carAdId)
            }
        }

        binding.toolbarCompareBtn.setOnClickListener {
            if (comparison){
                Utils.removeFromCompare(this, carAdId)
            }else{
                Utils.addToCompare(this, carAdId)
            }
        }

        binding.sellerProfileCv.setOnClickListener {
            val intent = Intent(this,
CarSellerProfileActivity::class.java)
            intent.putExtra("sellerUid", sellerUid)
            startActivity(intent)
        }

        binding.callBtn.setOnClickListener{
            Utils.callIntent(this, sellerPhone)
        }

        binding.smsBtn.setOnClickListener{
            Utils.smsIntent(this, sellerPhone)
        }



    }

    private fun editOptionsDialog(){
        Log.d(TAG, "editOptionsDialog: ")

        val  popupMenu = PopupMenu(this,
binding.toolbarEditBtn)

        popupMenu.menu.add(Menu.NONE, 0, 0,
"Змінити")
        popupMenu.menu.add(Menu.NONE, 1, 1,
"Позначити як проданий")

        popupMenu.show()

        popupMenu.setOnMenuItemClickListener {
menuItem ->
            val itemId = menuItem.itemId

            if (itemId == 0){
                val intent = Intent(this,
AdCarCreateActivity::class.java)
                intent.putExtra("isEditMode", true)
                intent.putExtra("carAdId", carAdId)
                startActivity(intent)
            }else if (itemId == 1){
                showMarkAsSoldDialog()

            }
            return@setOnMenuItemClickListener true
        }
    }

    private fun showMarkAsSoldDialog(){
        Log.d(TAG, "showMarkAsSoldDialog: ")

        val alertDialogBuilder =
MaterialAlertDialogBuilder(this)
        alertDialogBuilder.setTitle("Позначити як
проданий")
            .setMessage("Ви впевнені, що хочете
позначити цей автомобіль як проданий?")
            .setPositiveButton("Проданий"){dialog, which
->
                Log.d(TAG, "showMarkAsSoldDialog:
Натиснено проданий")

                val hashMap = HashMap<String, Any>()
                hashMap["status"] =
"${Utils.CAR_AD_STATUS_SOLD}"

                val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
                ref.child(carAdId)
                    .updateChildren(hashMap)
                    .addOnSuccessListener {
                        Log.d(TAG, "showMarkAsSoldDialog:
Marked as sold")
                    }
                    .addOnFailureListener {e ->
                        Log.e(TAG, "showMarkAsSoldDialog:
", e)
                        Utils.toast(this, "Не вдалося позначити
як проданий через ${e.message}")
                    }
            }
            .setNegativeButton("Скасувати"){dialog,
which ->
                Log.d(TAG, "showMarkAsSoldDialog:
Натиснено скасувати")
                dialog.dismiss()
            }
            .show()
    }

    private fun loadCarAdDetails(){
        Log.d(TAG, "loadCarAdDetails: ")

        val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
        ref.child(carAdId)
            .addValueEventListener(object :
```

```
ValueEventListener{
            override fun onDataChange(snapshot:
DataSnapshot) {
                try {
                    val modelCarAd =
snapshot.getValue(ModelCarAd::class.java)

                    sellerUid = "${modelCarAd!!.uid}"
                    val title = modelCarAd.title
                    val description =
modelCarAd.description
                    val year = modelCarAd.year
                    val status = modelCarAd.status
                    val condition = modelCarAd.condition
                    val model = modelCarAd.model
                    val brand = modelCarAd.brand
                    val price = modelCarAd.price
                    val location = modelCarAd.location
                    val timestamp = modelCarAd.timestamp

                    val formattedDate =
Utils.formatTimestampDate(timestamp)

                    if (sellerUid == firebaseAuth.uid){
                        binding.toolbarEditBtn.visibility =
View.VISIBLE
                        binding.toolbarDeleteBtn.visibility =
View.VISIBLE

                        binding.callBtn.visibility =
View.GONE
                        binding.smsBtn.visibility =
View.GONE


binding.sellerProfileLabelTv.visibility = View.GONE
                        binding.sellerProfileCv.visibility =
View.GONE
                    }else{
                        binding.toolbarEditBtn.visibility =
View.GONE
                        binding.toolbarDeleteBtn.visibility =
View.GONE

                        binding.callBtn.visibility
=View.VISIBLE
                        binding.smsBtn.visibility =
View.VISIBLE


binding.sellerProfileLabelTv.visibility =
View.VISIBLE
                        binding.sellerProfileCv.visibility =
View.VISIBLE
                    }

                    binding.titleTv.text = title
                    binding.descriptionTv.text = description
                    binding.yearTv.text = year
                    binding.statusTv.text = status
                    binding.conditionTv.text = condition
                    binding.modelTv.text = model
                    binding.brandTv.text = brand
                    binding.priceTv.text = price
                    binding.locationTv.text = location
                    binding.dateTv.text = formattedDate

                    loadSellerDetails()
                }catch (e: Exception){
                    Log.e(TAG, "onDataChange: ", e)
                }
            }

            override fun onCancelled(error:
DatabaseError) {

            }
        })
    }

    private fun loadSellerDetails(){
        Log.d(TAG, "loadSellerDetails: ")

        val ref =
FirebaseDatabase.getInstance().getReference("Users")
        ref.child(sellerUid)
            .addValueEventListener(object
:ValueEventListener{
                override fun onDataChange(snapshot:
DataSnapshot) {
                    val phoneCode =
"${snapshot.child("phoneCode").value}"
                    val phoneNumber =
"${snapshot.child("phoneNumber").value}"
                    val name =
"${snapshot.child("name").value}"
                    val profileImageUrl =
"${snapshot.child("profileImageUrl").value}"
                    val timestamp =
snapshot.child("timestamp").value as Long

                    val formattedDate =
Utils.formatTimestampDate(timestamp)

                    sellerPhone =
"$phoneCode$phoneNumber"

                    binding.sellerNameTv.text = name
                    binding.memberSinceTv.text =
formattedDate
                    try {
                        Glide.with(this@CarAdDetailsActivity)
                            .load(profileImageUrl)

.placeholder(R.drawable.ic_person_white)
                            .into(binding.sellerProfileIv)
```

```
                }catch (e: Exception){
                    Log.e(TAG, "onDataChange: ", e)
                }
            }

            override fun onCancelled(error:
DatabaseError) {

            }
        })
    }

    private fun checkIsFavorite(){
        Log.d(TAG, "checkIsFavorite: ")

        val ref =
FirebaseDatabase.getInstance().getReference("Users")

ref.child("${firebaseAuth.uid}").child("Favorites").chil
d(carAdId)
            .addValueEventListener(object :
ValueEventListener{
                override fun onDataChange(snapshot:
DataSnapshot) {
                    favorite = snapshot.exists()
                    Log.d(TAG, "onDataChange: favorite:
$favorite")

                    if (favorite){

binding.toolbarFavBtn.setImageResource(R.drawable.i
c_favorite_white)
                    }else{

binding.toolbarFavBtn.setImageResource(R.drawable.i
c_heart_white)
                    }
                }

                override fun onCancelled(error:
DatabaseError) {

                }
            })
    }

    private fun checkIsComparison(){
        Log.d(TAG, "checkIsComparison: ")

        val ref =
FirebaseDatabase.getInstance().getReference("Users")

ref.child("${firebaseAuth.uid}").child("Comparison").c
hild(carAdId)
            .addValueEventListener(object :
ValueEventListener{
                override fun onDataChange(snapshot:
DataSnapshot) {
                    comparison = snapshot.exists()
                    Log.d(TAG, "onDataChange: comparison:
$comparison")
                }

                override fun onCancelled(error:
DatabaseError) {

                }
            })
    }

    private fun loadCarAdImages(){
        Log.d(TAG, "loadCarAdImages: ")

        imageStringArrayList = ArrayList()

        val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
        ref.child(carAdId).child("Images")
            .addValueEventListener(object :
ValueEventListener{
                override fun onDataChange(snapshot:
DataSnapshot) {
                    imageStringArrayList.clear()

                    for (ds in snapshot.children){
                        try {
                            val modelImageSlider =
ds.getValue(ModelImageSlider::class.java)

imageStringArrayList.add(modelImageSlider!!)
                        }catch (e: Exception){
                            Log.e(TAG, "onDataChange: ", e)
                        }
                    }
                    val adapterImageSlider =
AdapterImageSlider(this@CarAdDetailsActivity,image
StringArrayList)
                    binding.imageSliderVp.adapter =
adapterImageSlider
                }

                override fun onCancelled(error:
DatabaseError) {

                }
            })
    }

    private fun deleteCarAd(){
        Log.d(TAG, "deleteCarAd: ")

        val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
        ref.child(carAdId)
```

```
            .removeValue()
            .addOnSuccessListener {
                Log.d(TAG, "deleteCarAd: Deleted")
                Utils.toast(this, "Видалено!")
                finish()
            }
            .addOnFailureListener { e ->
                Log.e(TAG, "deleteCarAd: ", e)
                Utils.toast(this, "Не вдалося видалити через
${e.message}")
            }
    }
}
```

```
package com.example.auto_finder.activities

import android.os.Bundle
import android.util.Log
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.bumptech.glide.Glide
import com.example.auto_finder.R
import com.example.auto_finder.Utils
import
com.example.auto_finder.adapters.AdapterCarAd
import
com.example.auto_finder.databinding.ActivityCarSelle
rProfileBinding
import com.example.auto_finder.models.ModelCarAd
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.FirebaseDatabase
import
com.google.firebase.database.ValueEventListener

class CarSellerProfileActivity : AppCompatActivity() {

    private lateinit var binding:
ActivityCarSellerProfileBinding

    private companion object{
        private const val TAG =
"SELLER_PROFILE_TAG"
    }

    private var sellerUid = ""

    override fun onCreate(savedInstanceState: Bundle?)
{
        super.onCreate(savedInstanceState)

        binding =
ActivityCarSellerProfileBinding.inflate(layoutInflater)
        setContentView(binding.root)


        sellerUid =
intent.getStringExtra("sellerUid").toString()

        Log.d(TAG, "onCreate: sellerUid: $sellerUid")

        loadSellerDetails()
        loadCarAds()

        binding.toolbarBackBtn.setOnClickListener {
            onBackPressed()
        }
    }

    private fun loadSellerDetails(){
        Log.d(TAG, "loadSellerDetails: ")

        val ref =
FirebaseDatabase.getInstance().getReference("Users")
        ref.child(sellerUid)
            .addValueEventListener(object :
ValueEventListener{
                override fun onDataChange(snapshot:
DataSnapshot) {
                    val name =
"${snapshot.child("name").value}"
                    val profileImageUrl =
"${snapshot.child("profileImageUrl").value}"
                    val timestamp =
snapshot.child("timestamp").value as Long

                    val formattedDate =
Utils.formatTimestampDate(timestamp)

                    binding.sellerNameTv.text = name
                    binding.sellerMemberSinceTv.text =
formattedDate
                    try {

Glide.with(this@CarSellerProfileActivity)
                            .load(profileImageUrl)

.placeholder(R.drawable.ic_person_white)
                            .into(binding.sellerProfileIv)
                    }catch (e: Exception){
                        Log.e(TAG, "onDataChange: ", e)
                    }
                }

                override fun onCancelled(error:
DatabaseError) {

                }
            })
    }

    private fun loadCarAds(){
        Log.d(TAG, "loadCarAds: ")

        val carAdArrayList: ArrayList<ModelCarAd> =
```

```
ArrayList()

        val ref =
FirebaseDatabase.getInstance().getReference("CarAds"
)
        ref.orderByChild("uid").equalTo(sellerUid)
            .addValueEventListener(object
:ValueEventListener{
                override fun onDataChange(snapshot:
DataSnapshot) {
                    carAdArrayList.clear()

                    for (ds in snapshot.children){
                        try {
                            val modelCarAd =
ds.getValue(ModelCarAd::class.java)

                            carAdArrayList.add(modelCarAd!!)
                        }catch (e: Exception){
                            Log.e(TAG, "onDataChange: ", e)
                        }
                    }
                    val adapterCarAd =
AdapterCarAd(this@CarSellerProfileActivity,
carAdArrayList)
                    binding.carAdsRv.adapter =
adapterCarAd

                    val carAdsCount =
"${carAdArrayList.size}"
                    binding.publishedAdsCountTv.text =
carAdsCount
                }

                override fun onCancelled(error:
DatabaseError) {

                }
            })
    }
}

package com.example.auto_finder.activities

import android.app.ProgressDialog
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.util.Patterns
import androidx.appcompat.app.AppCompatActivity
import com.example.auto_finder.Utils
import
com.example.auto_finder.databinding.ActivityLoginE
mailBinding
import com.google.firebase.auth.FirebaseAuth

class LoginEmailActivity : AppCompatActivity() {

    private lateinit var binding:
ActivityLoginEmailBinding

    private companion object{
        private const val TAG = "LOGIN_TAG"
    }

    private lateinit var firebaseAuth: FirebaseAuth

    private lateinit var progressDialog: ProgressDialog

    override fun onCreate(savedInstanceState: Bundle?)
{
        super.onCreate(savedInstanceState)
        binding =
ActivityLoginEmailBinding.inflate(layoutInflater)
        setContentView(binding.root)

        firebaseAuth = FirebaseAuth.getInstance()

        progressDialog = ProgressDialog(this)
        progressDialog.setTitle("Зачекайте.....")

progressDialog.setCanceledOnTouchOutside(false)

        binding.toolbarBackBtn.setOnClickListener {
            onBackPressed()
        }

        binding.noAccountTv.setOnClickListener {
            startActivity(Intent(this,
RegisterEmailActivity::class.java))
        }

        binding.loginBtn.setOnClickListener {
            validateData()
        }

    }
    private var email = ""
    private var password = ""

    private fun validateData(){
        email = binding.emailEt.text.toString().trim()
        password =
binding.passwordEt.text.toString().trim()

        Log.d(TAG, "validateData: email: $email")
        Log.d(TAG, "validateData: password:
$password")

        if
(!Patterns.EMAIL_ADDRESS.matcher(email).matches(
)){
            binding.emailEt.error = "Невірний формат
електронної пошти"
            binding.emailEt.requestFocus()
        }
        else if (password.isEmpty()){
```

```
            binding.passwordEt.error = "Введіть пароль"
            binding.passwordEt.requestFocus()
        }
        else{
            loginUser()
        }
    }

    private fun loginUser(){
        Log.d(TAG, "loginUser: ")
        progressDialog.setMessage("Авторизація")
        progressDialog.show()


firebaseAuth.signInWithEmailAndPassword(email,
password)
            .addOnSuccessListener {
                Log.d(TAG, "loginUser: Logged in...")
                progressDialog.dismiss()

                startActivity(Intent(this,
MainActivity::class.java))
                finishAffinity()
            }
            .addOnFailureListener{e ->
                Log.d(TAG, "loginUser: ", e)
                progressDialog.dismiss()

                Utils.toast(this, "Не можливо
авторизуватись через ${e.message}")
            }
    }
}
```

```
package com.example.auto_finder.activities

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import
com.example.auto_finder.databinding.ActivityLoginO
ptionsBinding

class LoginOptionsActivity : AppCompatActivity() {

    private lateinit var binding:
ActivityLoginOptionsBinding

    override fun onCreate(savedInstanceState: Bundle?)
{
        super.onCreate(savedInstanceState)

        binding =
ActivityLoginOptionsBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.closeBtn.setOnClickListener {
            onBackPressed()
        }
        binding.loginEmailBtn.setOnClickListener{
            startActivity(Intent(this,
LoginEmailActivity::class.java))
        }
    }
}
```