Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup

Виконав: студент 4 курсу
групи 4ПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Рудковський В.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф. ОТ Снігур А.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ д.т.н., проф. Романюк О.Н.
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«12» березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Рудковському Владиславу Вікторовичу

1. Тема роботи – Розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024.

3. Вихідні дані до роботи: тип програми – мобільний застосунок; модель розробки – ітеративна; засоби зберігання даних – Firebase; вхідні дані: облікові дані користувача, оголошення та інформація про транспортні засоби; вихідні дані: інформація у вигляді оголошень; середовище розробки – Android Studio; мова програмування – Kotlin; програмне забезпечення для симуляції мобільних пристроїв – Android Emulator.

4. Зміст текстової частини: вступ, обґрунтування вибору методу розробки та постанова задач дослідження, розробка структури та алгоритмів роботи програмного продукту, розробка мобільного застосунку для купівлю та продажу автомобілів, тестування програмного застосунку, висновки, список використаних джерел, додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; мета, об'єкт та предмет дослідження; задачі дослідження; новизна і практична цінність одержаних результатів; порівняльний аналіз аналогів; архітектура мобільного застосунку; загальний алгоритм роботи системи; розробка алгоритму для збереження вподобаних оголошень; демонстрація інтерфейсу застосунку «Початковий екран»; демонстрація інтерфейсу застосунку «Екран реєстрації та

авторизації»; демонстрація інтерфейсу застосунку «Головний екран» демонстрація інтерфейсу застосунку «Створення оголошення»; демонстрація інтерфейсу застосунку «Збереження оголошень»; Тестування валідності паролю та електронної пошти; Тестування алгоритму створення оголошень; Апробація та публікація матеріалів бакалаврської дипломної роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата			
		Завдання видав		Завдання прийняв	
1-4	Катєльніков Д.І. к.т.н., доцент кафедри ПЗ	13.03.24		03.06.24	

7. Дата видачі завдання 13 березня 2024 р

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз застосунку для пошуку, купівлі та продажу автомобілів та вибір цілей для поставленої задачі	13.03.24 – 20.03.24	вик
2	Розробка алгоритмів застосунку	13.03.24 – 20.03.24	вик
3	Розробка алгоритму для збереження вподобаних оголошень	22.03.24 – 13.04.24	вик
4	Розробка реєстрації та авторизації	15.04.24 – 25.04.24	вик
5	Розробка валідації реєстрації	26.04.24 – 03.05.24	вик
6	Розробка бази даних застосунку	04.05.24 – 11.05.24	вик
7	Тестування програмного забезпечення	12.05.24 – 23.05.24	вик
8	Оформлення матеріалів до захисту БДР	25.05.24 – 03.06.24	вик

Студент _____ (підпис) **В. В. Рудковський** (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ (підпис) **Д. І. Катєльніков** (ініціали та прізвище)

АНОТАЦІЯ

УДК 339.13.017+629.331:(004.4)

Рудковський В.В. Розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup: бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2024. 60 с.

На укр. мові. Бібліогр. : 25 назв.; рис. : 19; табл. : 10.

У бакалаврській дипломній роботі було визначено доцільність, практичну цінність та проведено детальний аналіз програмного мобільного застосунку для купівлі-продажу автомобілів.

Запропоновано модифікацію алгоритму розміщення оголошень що буде орієнтований на зручне та швидке розміщення оголошень. Розроблено аглоритми реєстрації та авторизації, збереження вподобаних оголошень та валідації реєстрації.

Програмний застосунок розроблено із використанням середовища розробки Android Studio та мови програмування Kotlin. Під час розробки було використано Firebase та бібліотеку jsoup. У результаті виконання бакалаврської дипломної роботи розроблено мобільний застосунок, ефективність якого перевірено та підготовлено інструкцію користувача.

Отримані в бакалаврській дипломній роботі результати можна використати для пошуку, купівлі та продажу автомобілів.

Ключові слова: купівля автомобіля, продаж автомобіля, відсутність повної інформації, недостовірність інформації, мобільний додаток.

ABSTRACT

UDC 339.13.017+629.331:(004.4)

Rudkovsky V.V. Development of a personal consultant mobile application for finding, buying and selling cars using Kotlin, Firebase and jsoup library. Vinnytsia: VNTU, 2024. 60 p.

In Ukrainian language. Bibliobrapher: 25 titles; fig. : 19; tabl. : 10.

The bachelor's thesis determined the feasibility, practical value and carried out a detailed analysis of the software mobile application for buying and selling cars.

A modification of the ad placement algorithm is proposed, which will be focused on convenient and fast ad placement. Aglorhythms for registration and authorization, saving of liked ads and validation of registration have been developed.

The software application is developed using the Android Studio development environment and the Kotlin programming language. Firebase and the jsoup library were used during development. As a result of completing the bachelor's thesis, a mobile application was developed, the effectiveness of which was checked and a user manual was prepared.

The results obtained in the bachelor thesis can be used for searching, buying and selling cars.

Keywords: buying a car, selling a car, lack of complete information, unreliability of information, mobile application.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний світ неможливо уявити без автомобілів, які стали не лише зручним засобом пересування, а й важливим елементом соціального, економічного та культурного життя. Автомобільний ринок все швидше розвивається та стає більш конкурентним, постійно зростає кількість автомобільних брендів, моделей та характеристик що призводить до ускладнення вибору оптимальної моделі авто для людини зважаючи на її потреби та бюджет.

Без мобільних застосунків, пошук та купівля автомобілів стає значно складнішим та затратним процесом, що призводить до марно витраченого часу. Багато хто користується мобільними застосунками для пошуку та купівлі автомобілів через їх зручності та доступності. Вони дозволяють користувачам швидко та ефективно проглядати різноманітні автомобільні пропозиції, порівнювати ціни та характеристики, а також здійснювати важливі функції, такі як зв'язок з продавцями, планування оглядів та тест-драйву.

Мобільні застосунки також надають користувачам можливість отримувати повідомлення про нові автомобільні пропозиції, акції та знижки, що дозволяє їм залишатися в курсі подій на ринку автомобілів.

Завдяки мобільним застосункам, пошук та купівля автомобіля стають більш зручними, ефективними та приємними процесами для широкого кола користувачів. Такий підхід дозволяє покупцям швидше знаходити відповідні автомобільні пропозиції та здійснювати обмін інформацією з продавцями, що сприяє зниженню загальних витрат часу та зусиль.

На сьогоднішній день вже існують мобільні застосунки для продажу автомобілів, але вони мають свої недоліки, що варто враховувати. Першим недоліком таких застосунків є недостатньо актуальна інформація про автомобілі та їх характеристики, що виникає через не вчасне оновлення інформації або через помилки в базі даних. Проблема обмеженості функціоналу є достатньо важливою, адже вони можуть бути обмеженими в можливостях, наприклад, вони

можуть не мати важливих функцій, таких як можливість порівняння автомобілів або отримання історії обслуговування та ДТП. Крім того, якщо застосунок має низьку репутацію або виникають випадки шахрайства, то це призводить до недовіри користувачів, що у свою чергу веде до зниження активності користувачів. Тому розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів все ще є актуальною задачею.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є підвищення ефективності операцій пошуку, купівлі та продажу автомобіля шляхом розробки мобільного застосунку персонального консультанта, який дозволить оперативно оновлювати інформацію, фільтрувати, надавати рекомендації та підтримувати комунікацію з контрагентами.

Основними **задачами дослідження** є:

- розробити графічний інтерфейс;
- розробити модуль розміщення оголошень;
- розробити модуль авторизації та реєстрації користувачів;
- розробити модуль збереження вподобаних оголошень;
- розробити модуль валідації реєстрації*;
- розробити метод збереження зареєстрованих користувачів до бази даних Firebase;
- розробити систему зв'язку між покупцями та продавцями;
- провести тестування мобільного застосунку персонального консультанта.

Об'єкт дослідження – процес пошуку, купівлі та продажу автомобілів за допомогою мобільних застосунків.

Предметом дослідження є засоби реалізації мобільної застосунку персонального консультанта, орієнтованого на покращення досвіду корситувача

у купівлі та продажу авто.

Методи дослідження: Протягом дослідження було використано: теорію людино-машинної взаємодії для побудови інтерфейсу, теорію баз даних для організації зберігання інформації, методи об’єктно-орієнтованого програмування для розробки архітектури класів та модулів програмного забезпечення та теорію розподілених систем для організації зв’язку із сервером підтримки активності.

Новизна отриманих результатів:

Подальшого розвитку отримав метод представлення інформації про авторинок, у якому на відміну від існуючих методів представлення інформації, комбінується інформація про наявні на поточний момент авто, яка оперативно оновлюється, та параметри вподобаних авто, які накопичуються з часом, дозволяючи користувачу формувати авто своєї мрії.

Практична цінність отриманих результатів. Практична цінність полягає у розробці мобільного застосунку персонального консультанта, який можливо використовувати для пошуку, купівлі та продажу автомобілей.

Особистий внесок здобувача. Усі наукові результати, що викладені у бакалаврській дипломній роботі, отримано автором самостійно.

Апробація результатів роботи. Основні положення бакалаврської дипломної роботи доповідалися на Всеукраїнській науково-технічній конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024). Секція Інформаційні технології та комп‘ютерна інженерія».

Публікації. За тематикою дослідження опубліковано одну наукову працю у збірниках матеріалів конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз мобільних застосунків для пошуку, купівлі та продажу автомобілів

Мобільні застосунки для пошуку, купівлі та продажу автомобілів стають все більш популярними [1], пропонуючи користувачам зручний та гнучкий спосіб знайти та придбати транспортний засіб. Ці застосунки пропонують широкий спектр функцій, включаючи:

– пошук автомобілів за різними критеріями, такими як марка, модель, рік випуску, ціна, пробіг та функції;

– перегляд фотографій та описів автомобілів;

– зв'язок з продавцями для отримання додаткової інформації або організації тест-драйву;

– подання заявок на фінансування або лізинг;

– завершення покупки автомобіля безпосередньо через програму.

Існує багато переваг використання мобільних застосунків для пошуку, купівлі та продажу автомобілів, включаючи:

– зручність. Ці системи доступні на будь-якому мобільному пристрої, що дозволяє користувачам шукати та купувати автомобілі з будь-якого місця та в будь-який час;

– швидкість. Мобільні системи можуть допомогти користувачам швидко знайти автомобілі, які відповідають їхнім потребам та бюджету;

– прозорість. Ці застосунки можуть надавати користувачам доступ до широкого кола інформації про автомобілі, включаючи їх історію, ціни та відгуки;

– економія часу та грошей. Мобільні системи можуть допомогти користувачам заощадити час та гроші, полегшуючи процес пошуку, купівлі та продажу автомобілів.

Ринок мобільних застосунків для пошуку, купівлі та продажу автомобілів швидко зростає. За оцінками експертів, до 2025 року ринок сягне $20 мільярдів.

Існує кілька ключових тенденцій, які формують майбутнє ринку мобільних застосунків для пошуку, купівлі та продажу автомобілів. Ці тенденції включають:

– зростання популярності онлайн-покупок. Все більше людей купують автомобілі онлайн, що призводить до зростання попиту на мобільні застосунки персональних консультантів;

– використання штучного інтелекту. Штучний інтелект використовується для покращення можливостей мобільних застосунків, таких як персоналізація результатів пошуку та надання рекомендацій;

– персоналізація. Мобільні системи стають все більш персоналізованими, пропонуючи користувачам досвід, який відповідає їхнім індивідуальним потребам;

– інтеграція з іншими платформами. Мобільні системи інтегруються з іншими платформами, такими як фінансові установи та страхові компанії, щоб зробити процес покупки автомобіля більш плавним.

Отже, Мобільні застосунки для пошуку, купівлі та продажу автомобілів стають все більш важливим інструментом для покупців та продавців автомобілів. Ці системи пропонують ряд переваг, включаючи зручність, швидкість, прозорість та економію часу та грошей.

1.2 Порівняльний аналіз аналогів

Існує багато застосунків консультантів, які допомагають людям шукати, продавати та купувати автомобілі. Ці системи пропонують різні функції та послуги, тому важливо вибрати ту, яка найкраще відповідає вашим потребам.

OLX – це популярний онлайн-майданчик для оголошень, де можна знайти широкий спектр автомобілів (див. рисунок 1.1). OLX надає можливість розміщення безкоштовних оголошень для приватних користувачів та платних пакетів послуг для бізнес-користувачів [2], що дозволяє розміщувати оголошення з більш високою видимістю. Крім того, OLX забезпечує широкий спектр категорій та фільтрів для зручного пошуку, а також можливість

використання різних мов і регіональних версій, що робить платформу доступною для користувачів з різних країн.

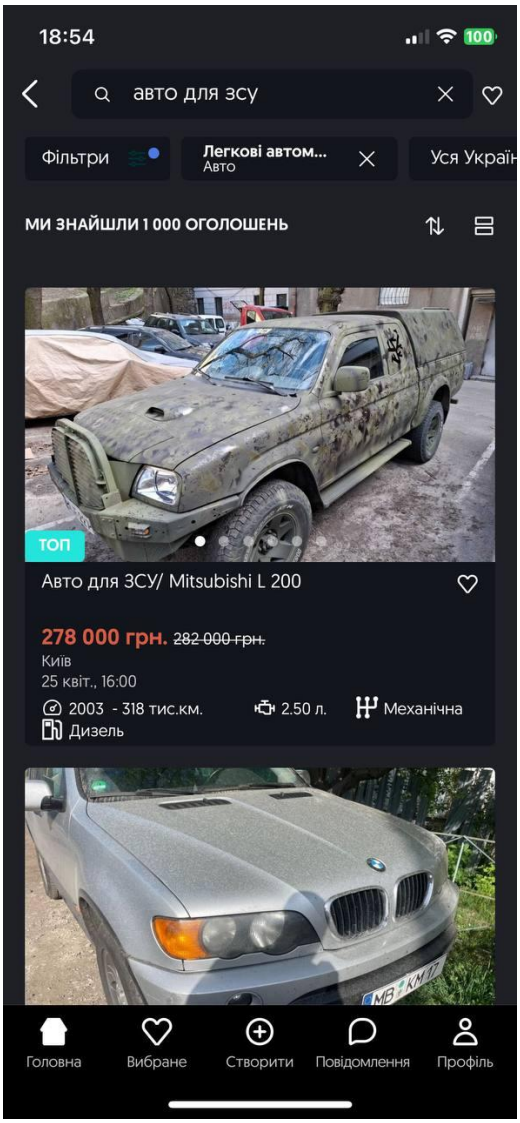

Рисунок 1.1. – Зображення програмного продукту «OLX»

Переваги OLX:

– широке охоплення аудиторії: OLX є одним з найпопулярніших онлайн-платформ для купівлі та продажу товарів, включаючи автомобілі, що забезпечує велику аудиторію потенційних покупців;

– безкоштовні оголошення: OLX дозволяє користувачам розміщувати оголошення про продаж безкоштовно, що дозволяє економити кошти на рекламі;

– різноманітність автомобільного асортименту: Платформа пропонує великий вибір автомобілів різних марок, моделей, років випуску та цінових категорій;

– можливість перевірки історії авто: Деякі продавці можуть надавати інформацію про історію обслуговування, технічний стан та інші важливі деталі щодо автомобіля;

– простота використання: Веб-сайт та мобільний застосунок OLX мають інтуїтивно зрозумілий і простий інтерфейс, що полегшує розміщення оголошень та пошук автомобілів.

Недоліки OLX:

– велика конкуренція: Завдяки широкому колу користувачів OLX, конкуренція серед продавців автомобілів може бути високою, що ускладнює продаж;

– ризик шахрайства та обману: На OLX можуть з'являтися недобросовісні продавці або оголошення з фальшивою інформацією, що може призвести до обману або купівлі автомобіля з проблемами;

– необхідність самостійного організації перевірки автомобіля: OLX не надає послуги перевірки технічного стану або історії автомобіля, тому покупець повинен самостійно організовувати перевірку перед покупкою;

– обмежена можливість довіри до продавця: На відміну від дилерських центрів або платформ з гарантіями, які надають певний рівень довіри для покупців, на OLX довіра до продавців може бути обмеженою;

– відсутність гарантій та підтримки покупців: OLX не надає гарантій на куплені автомобілі та не забезпечує підтримку покупцям у вирішенні спірних питань або проблем з транзакціями.

Також у OLX є недоліки, що призводять до того, що оголошення не завжди є достовірними, а відсутність можливості перевірки історії автомобіля може викликати непевність у покупців щодо стану та якості автомобілів, що розміщуються на платформі. Також, як і на будь-якому онлайн-майданчику,

існує ризик шахрайства, який може відлякати деяких користувачів від використання платформи.

Auto.ria – це один з найбільших та найпопулярніших онлайн-платформ в Україні для продажу та покупки автомобілів [3]. Платформа пропонує широкий вибір автомобілів різних марок, моделей, років випуску та цінових категорій. Користувачі можуть знайти як нові, так і вживані автомобілі в різному стані та конфігураціях. Auto.ria має велику базу даних, що дозволяє користувачам знайти авто за будь-якими критеріями. Система фільтрації дозволяє точно налаштувати свої критерії пошуку, щоб знайти відповідні автомобілі. Багато оголошень містять детальні технічні характеристики, фотографії та історію обслуговування автомобіля. Користувачі можуть перевірити історію транспортного засобу за допомогою VIN-коду, що забезпечує більшу впевненість у покупця. (див. рисунок 1.2). Переваги: великий вибір, зручний пошук, можливість перевірити історію автомобіля.

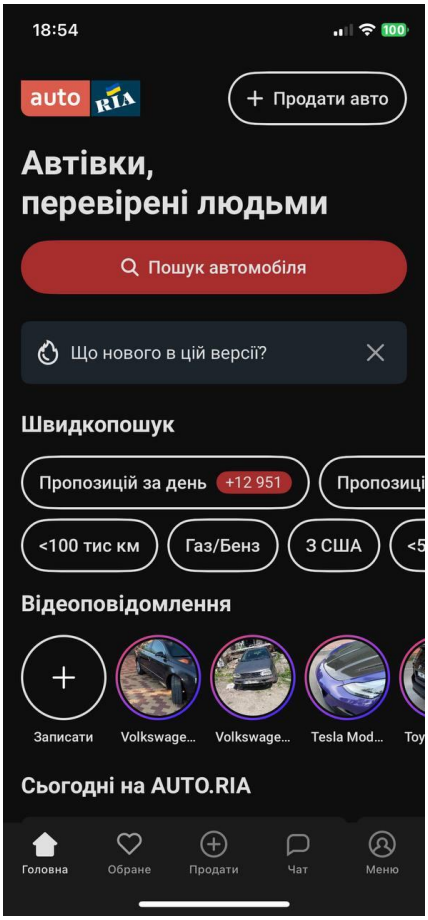

Рисунок 1.2. – Зображення програмного продукту «Autoria»

Переваги Autoria:

– довіра та безпека: Autoria є відомою та довіреною платформою для продажу автомобілів, що забезпечує більш високий рівень безпеки для покупців і продавців;

– комплексність інформації: Autoria зазвичай надає більш детальну інформацію про автомобілі, включаючи технічний стан, історію обслуговування, рейтинги якості та відгуки користувачів;

– гарантія якості: Деякі автомобілі на Autoria можуть бути піддані сертифікації або перевірці на відповідність стандартам якості, що забезпечує покупцям більш високий рівень впевненості;

– підтримка клієнтів: Платформа може надавати підтримку клієнтам у вирішенні спірних ситуацій, надавати консультації щодо вибору автомобіля та інші послуги;

– широкий вибір автомобілів: Autoria пропонує великий асортимент автомобілів різних марок, моделей, років випуску та цінових категорій.

Недоліки Autoria:

– обмежена аудиторія: Autoria може мати меншу аудиторію порівняно з загальнодоступними платформами, що може обмежити кількість потенційних покупців;

– вартість розміщення оголошень: Деякі види оголошень на Autoria можуть бути платними, що може створювати додаткові витрати для продавців;

– менша ринкова присутність: У деяких регіонах або країнах Autoria може мати меншу ринкову присутність порівняно з іншими платформами;

– необхідність перевірки інформації: Інформація про автомобілі на Autoria може бути неперевіреною або неповною, тому покупцеві слід уважно перевіряти всю інформацію перед покупкою;

– можливість шахрайства: Хоча Autoria зазвичай забезпечує більший рівень безпеки, існує ризик зустрічі з недобросовісними продавцями або оголошеннями з фальшивою інформацією.

Auto.ria має платні пакети послуг, які можуть обмежувати доступ до деяких функцій для деяких користувачів [4]. Це може створювати додаткові витрати для тих, хто шукає безкоштовні або доступні для всіх рішення. Крім того, незважаючи на те, що платформа надає можливість перевірки історії автомобіля за допомогою VIN-коду, інформація про автомобіль може бути не завжди повною або точною, що може викликати недовіру у деяких користувачів. Також, не виключена можливість виникнення проблем з оголошеннями або сервісом платформи, що може призвести до нестабільності або несприятливого досвіду користувача.

Avtomoto.ua – це мобільний застосунок, який пропонує інформацію про нові та вживані автомобілі, а також послуги з продажу та купівлі автомобілів (див. рисунок 1.3). Переваги: широка база даних автомобілів, можливість порівняти ціни, допомога з оформленням документів.

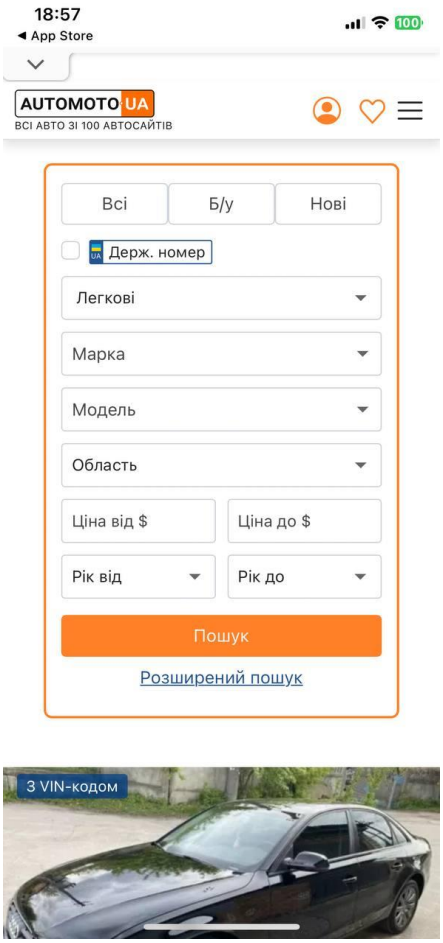

Рисунок 1.3 – Зображення програмного продукту «avtomoto»

Переваги Avtomoto.ua:

– широкий вибір автомобілів: Avtomoto.ua пропонує різноманітний асортимент автомобілів різних марок, моделей, років випуску та цінових категорій, що задовольняє різні потреби покупців;

– детальна інформація: Платформа зазвичай надає докладну інформацію про автомобілі, включаючи технічний стан, історію обслуговування, фотографії та відгуки користувачів;

– можливість порівняння: Avtomoto.ua дозволяє користувачам порівнювати різні автомобілі за різними критеріями, що сприяє у виборі найбільш підходящої опції;

– безпека та довіра: Як довірена платформа, Avtomoto.ua забезпечує високий рівень безпеки для покупців і продавців, що створює сприятливу атмосферу для проведення угод;

– інтуїтивний інтерфейс: Веб-сайт та мобільний додаток Avtomoto.ua мають зручний та легкий у використанні інтерфейс, що полегшує пошук та розміщення оголошень.

Недоліки Avtomoto.ua:

– обмежена аудиторія: У деяких регіонах або країнах Avtomoto.ua може мати обмежену аудиторію порівняно з іншими популярними платформами;

– вартість розміщення оголошень: Деякі види оголошень на Avtomoto.ua можуть бути платними, що може вплинути на витрати продавців;

– необхідність перевірки інформації: Інформація про автомобілі може бути неперевіреною або неповною, тому покупцям слід уважно перевіряти всю інформацію перед покупкою;

– конкуренція: У зв'язку з популярністю Avtomoto.ua серед продавців автомобілів може бути висока конкуренція, що може ускладнити продаж;

– можливість шахрайства: Існує ризик зустрічі з недобросовісними продавцями або оголошеннями з фальшивою інформацією, хоча платформа зазвичай забезпечує високий рівень безпеки.

Також недоліками Avtomoto.ua можуть бути обмежені можливості функціоналу порівняно з іншими конкуруючими платформами. Наприклад, може бути менш розвинута система фільтрації або менше додаткових сервісів для користувачів. Також, як і на будь-якому онлайн-платформі, існує ризик зустрічі з неякісними оголошеннями або можливістю шахрайства. Крім того, доступність певних функцій або послуг може бути обмежена залежно від регіону або типу користувача, що може вплинути на загальний досвід користувача платформою.

Основними перевагами створення власної реалізації мобільного застосунку для пошуку, продажі та купівлі автомобілів є:

– об'єднання всіх функцій. Нова система може об'єднати всі найкращі функції існуючих систем, пропонуючи більш комплексне та зручне рішення;

– персоналізація. Нова система може бути персоналізована для потреб кожного користувача, пропонуючи їм більш релевантні результати та рекомендації;

– інновації. Нова система може використовувати інноваційні технології, такі як штучний інтелект та машинне навчання, щоб забезпечити кращий досвід для користувачів;

– конкурентоспроможні ціни. Нова система може пропонувати більш конкурентоспроможні ціни, ніж існуючі системи.

Аргументи на користь створення нової системи-консультанта:

– зростаючий ринок. Ринок автомобілів постійно зростає, тому існує великий потенціал для успіху нової системи-консультанта;

– незадоволеність користувачів. Існуючі застосунки не завжди задовольняють потреби користувачів, тому існує попит на новий, кращий застосунок;

– технологічний прогрес. Технологічний прогрес створює нові можливості для розробки більш досконалих мобільних застосунків.

Результати порівняння аналогів наведено в табл. 1.1.

Таблиця 1.1. – Порівняльні характеристики програмних продуктів

Система	Переваги	Недоліки
OLX	Великий вибір, зручний пошук, можливість спілкуватися з продавцями безпосередньо	Багато шахраїв, не завжди є можливість перевірити історію автомобіля
Auto.ria	Великий вибір, зручний пошук, можливість перевірити історію автомобіля	Платний доступ до деяких функцій
Avtomoto.ua	Широка база даних автомобілів, можливість порівняти ціни, допомога з оформленням документів	Не такий великий вибір, як на OLX та Auto.ria
Нова система	Об'єднання всіх функцій, персоналізація, інновації, конкурентоспроможні ціни	Імовірність шахрайства

Отже, створення мобільного застосунку персонального консультанта для пошуку, продажу та купівлі автомобілів є цілком актуальним в наш час. Новий застосунок може об'єднати всі найкращі функції існуючих мобільних систем, пропонуючи більш комплексне та зручне рішення. Її можна персоналізувати для потреб кожного користувача, пропонуючи їм більш релевантні результати та рекомендації.

1.3 Аналіз методів розв’язання поставленої задачі

Для розробки мобільного застосунку для пошуку, продажу та купівлі автомобілів можна розглянути декілька методів [5]. Ось деякі з них:

1. Метод інтерфейсу користувача (UI/UX):

– персоналізований дизайну. Система повинна мати зручний та привабливий інтерфейс, який привертає користувачів та робить навігацію простою та інтуїтивно зрозумілою;

– мобільна дружність. Враховуючи зростаючу популярність мобільних пристроїв, система повинна бути оптимізована для роботи на різних пристроях та розмірах екранів.

2. Метод аналізу даних та рекомендацій:

– аналіз поведінки користувача. Збір та аналіз даних про перегляди, пошуки та вибори автомобілів користувачами допомагає створити персоналізовані рекомендації;

– машинне навчання та алгоритми рекомендацій [6]. Використання алгоритмів машинного навчання для аналізу даних та надання користувачам рекомендацій щодо автомобілів, які відповідають їхнім потребам та вподобанням.

3. Метод інтеграції з базами даних автомобілів та дилерськими мережами:

– підключення до баз даних автомобілів. Інтеграція з базами даних, які містять інформацію про різні марки, моделі та технічні характеристики автомобілів;

– співпраця з дилерськими мережами. Укладення партнерських угод з авто дилерами для отримання доступу до їхніх пропозицій та інформації про наявні автомобілі.

4. Метод інтеграції з платформами онлайн-торгівлі та платіжними системами:

– підключення до онлайн-платформ [7]. Інтеграція з популярними онлайн-платформами для торгівлі автомобілями дозволяє користувачам отримувати доступ до широкого асортименту автомобілів;

– забезпечення безпеки платежів. Інтеграція з безпечними платіжними системами для забезпечення безпеки та зручності платежів під час покупки автомобілів онлайн.

5. Метод інтеграції з GPS та картографічними сервісами:

– визначення місця розташування: Інтеграція з GPS дозволяє користувачам швидко знаходити автомобілі в їхній регіоні;

– відображення на мапі: Інтеграція з картографічними сервісами дозволяє відображати розташування автомобілів на мапі для зручності користувачів.

Ці методи можна поєднати для створення комплексного та ефективного мобільного застосунку персонального консультанта для пошуку, продажу та купівлі автомобілів. Ключовим аспектом буде розуміння потреб та вимог користувачів, а також постійне вдосконалення системи на основі отриманих результатів та змін на ринку.

1.4 Постановка задач для розробки мобільного застосунку для пошуку, купівлі та продажу авто

Проаналізувавши переваги та недоліки існуючих систем було визначено наступні завдання, які необхідно виконати для розробки мобільного застосунку:

– розробити графічний інтерфейс;

– розробити модуль розміщення оголошень;

– розробити модуль авторизації та реєстрації користувачів;

– розробити модуль збереження вподобаних оголошень;

– розробити модуль валідації пошти та паролю;

– розробити метод збереження зареєстрованих користувачів до бази даних Firebase;

– розробити систему зв’язку між покупцями та продавцями;

– провести тестування мобільного застосунку персонального консультанта.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі роботи було проведено докладне обстеження наявних мобільних застосунків для пошуку та продажу автомобілів, зокрема, таких як OLX, Autoria та automoto. Це включало в себе аналіз їхніх функцій, можливостей, інтерфейсу та загального враження від користування.

В результаті аналізу було виявлено, що незважаючи на існуючі аналоги, існує попит на новий, більш інноваційний мобільний застосунок, який зможе задовольнити потреби користувачів більш ефективно та зручно. Порівняльний аналіз дозволив виявити слабкі та сильні сторони існуючих рішень, що забезпечить змогу врахувати їх при розробці нового мобільного застосунку. Було сформульовано задачі, які необхідно виконати для розробки системи.

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних та вихідних даних системи

Вхідні дані представляють собою інформацію, яка вводиться користувачем або надходить з інших джерел до системи. Вони необхідні для правильної роботи мобільного застосунку чи для обробки в рамках певного процесу.

Розробка програмних компонентів для мобільного застосунку у сфері авто включає кілька важливих етапів. Одним із головних є ретельний аналіз логіки роботи системи та визначення основного функціоналу можливостей.

Насамперед важливим кроком є перша реєстрація, у випаду якщо користувач не зареєстрований у мобільному застосунку, це і будуть перші вхідні дані що введені за допомогою графічного інтерфейсу. Після виконання етапу реєстрації дані будуть заносить у базу даних. Це надає безпеку даних користувачів, забезпечуючи доступ до особистих кабінетів тільки тим, кому вони належать.

Інтерфейс грає важливу роль у взаємодії користувача з програмним застосунком і визначає зручність та ефективність використання застосунку. Тому його розробка потребує ретельної уваги до потреб користувачів та дотримання сучасних дизайнерських рішень.

Також, не менш важлива інформація яку користувач буде додавати в оголошення для продажу автомобілів. У користувачів буде можливість створювати свої оголошення для продажу автомобілів та можливість перегляду оголошень автомобілів з використанням різноманітних фільтрів. А також буде можливість зберігати вподобані оголошення. Для цього необхідно вказати назву автомобіля, його марку, повну характеристику, додати зображення.

До вихідних даних мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів можна віднести створені оголошення автомобілів, які користувач зможе переглядати, фільтрувати та сортувати.

Для реалізації мобільного застосунку для пошуку, купівлі та продажу автомобілів будуть використані засоби програмування Kotlin та інтегроване середовище Android Studio. Android Studio є основним інструментом для розробки Android-додатків та надає потужні засоби для створення високоякісних та функціональних програм для мобільних пристроїв [8]. Kotlin – це мова програмування, що надає зручний синтаксис , безпеку та надійність, що робить її привабливим вибором для розробки широкого спектру програм на Android [9].

2.2 Розробка структури мобільного застосунку

Створення структури мобільного застосунку є досить тривалим та складним процесом, що включає в себе проектування та організацію всіх складових його функціоналу для оптимальної роботи на мобільних пристроях. Це охоплює визначення можливостей, взаємодію з користувачем, структуру даних, навігацію між екранами. Архітектура MVVM (Model-View-ViewModel) стала популярним підходом для розробки Android-додатків, оскільки вона дозволяє ефективно розділити компоненти застосунку на три основні частини такі як модель (Model), вид (View) та ViewModel. Модель відповідає за управління даними та бізнес-логікою, вид за представлення інформації користувачу, а ViewModel за підготовку та управління даними для відображення у виді.

Основні частини MVVM:

– view. Це опис інтерфейсу користувача компонентів або сторінок. У цьому шарі відображається інформація та анімація, а також пропонується взаємодія з користувачами. Він просто прив’язаний до ViewModel, що означає, що ViewModel є унікальним джерелом даних View – дані включають текст, розмір, зображення тощо – і операції користувачів також запускають подію ViewModel. Таким чином, View робить лише дві речі, оголошуючи компоненти та призначаючи їх джерело даних і подію;

– viewModel. Властивості ViewModel завжди є динамічними даними, що означає, що вони часто змінюються. Отже, потрібно написати метод

встановлення властивості. Поки властивість встановлено, наприклад, через відповідь мережі на рівні моделі, спостерігачу буде опубліковано сигнал, який використовується для автоматичного оновлення відповідних даних View;

– model. Це абстрактна модель для даних, не пов’язаних з інтерфейсом користувача, хоча, можливо, деякі поля такі ж, як ViewModel. Крім того, на цьому рівні можуть бути записані деякі фундаментальні процеси обробки даних і бізнес-логіка, наприклад мережевий запит, робота з базою даних, зв’язок із сервером підтримки активності.

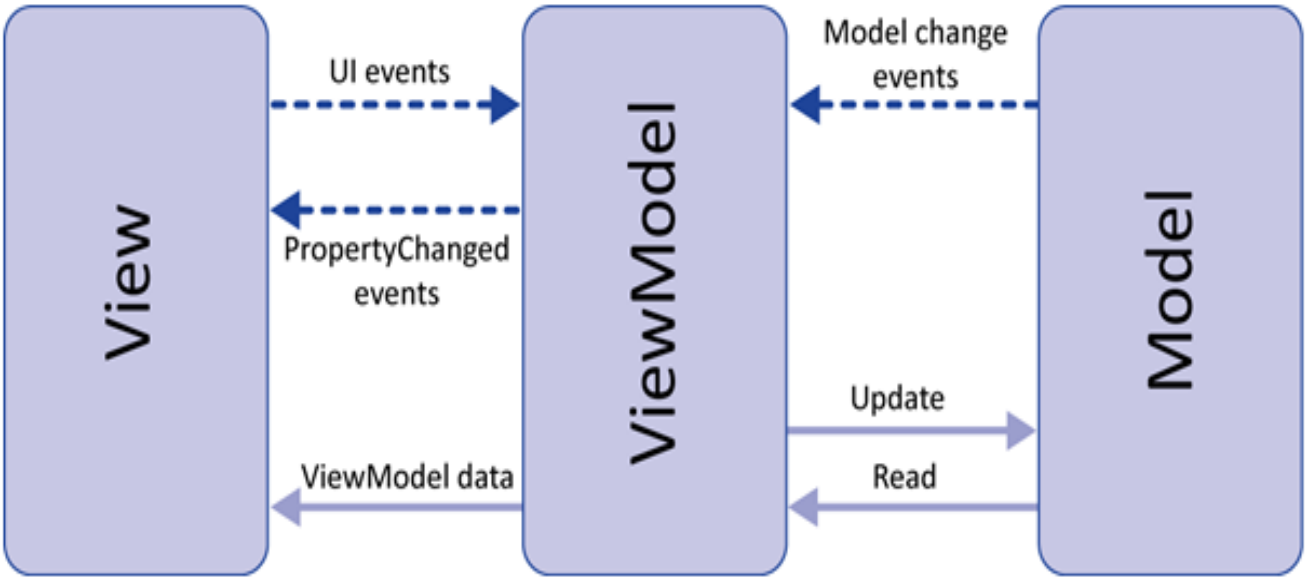

Рисунок 2.1 – Схема архітектури MVVM

Використання MVVM на Android має кілька переваг, включаючи спрощення процесу тестування, зменшення взаємозалежності компонентів, поліпшення читабельності коду та підтримку асинхронного програмування.

Основними принципами MVVM є двостороннє зв'язування даних що забезпечує автоматичне оновлення інтерфейсу користувача при зміні даних у ViewModel і навпаки. Завдяки цьому, Вигляд завжди відображає актуальні дані, а зміни, внесені користувачем, автоматично передаються у ViewModel. Також команди “Commands” використовуються для обробки дій користувача, таких як натискання кнопок.

Переваги використання MVVM:

– спрощення процесу тестування. MVVM розділяє компоненти додатку на логічні частини, що полегшує тестування кожної з них окремо. ViewModel, яка містить бізнес-логіку, може бути легко протестована без необхідності запуску всього UI. Це дозволяє писати автоматизовані тести для ViewModel, що полегшує виявлення та виправлення помилок;

– зменшення взаємозалежності компонентів. MVVM допомагає уникнути прямих зв'язків між видом (View) і моделлю (Model), що робить код більш масштабованим і підтримуваним. ViewModel дозволяє ізолювати бізнес-логіку від UI-коду, що допомагає зберегти його чистим і організованим;

– поліпшення читабельності коду. Розділення компонентів додатку за допомогою MVVM робить код більш зрозумілим. Кожен компонент відповідає за свою конкретну відповідальність, що спрощує розуміння функціональності і структури програми;

– підтримка асинхронного програмування. ViewModel може легко обробляти асинхронні операції, такі як мережеві запити або завантаження даних з бази даних, за допомогою LiveData або RxJava. Це дозволяє реалізувати ефективну обробку даних у додатку, що покращує користувацький досвід.

2.3 Розробка загального алгоритму роботи програми

Для кращого розуміння роботи мобільного застосунку для пошуку, купівлі та продажу автомобілів було вирішено розробити модель роботи системи на прикладі діаграми (див. рисунок 2.2), [10]. Згідно вказаної діаграми користувач повинен пройти реєстрацію з вибором одного із запропонованих способів, у випадку якщо користувач вже зареєстрований, то відкривається вікно авторизації. Після входу в особистий аккаунт одразу доступна можливість перегляду оголошень автомобілів, якщо користувача влаштовує авто то в нього буде можливість зв’язатися з автором оголошення для уточнення даних про авто та обговорення самої покупки. Також присутні можливість створення оголошень з повним внесенням даних про автомобілі до бази даних.

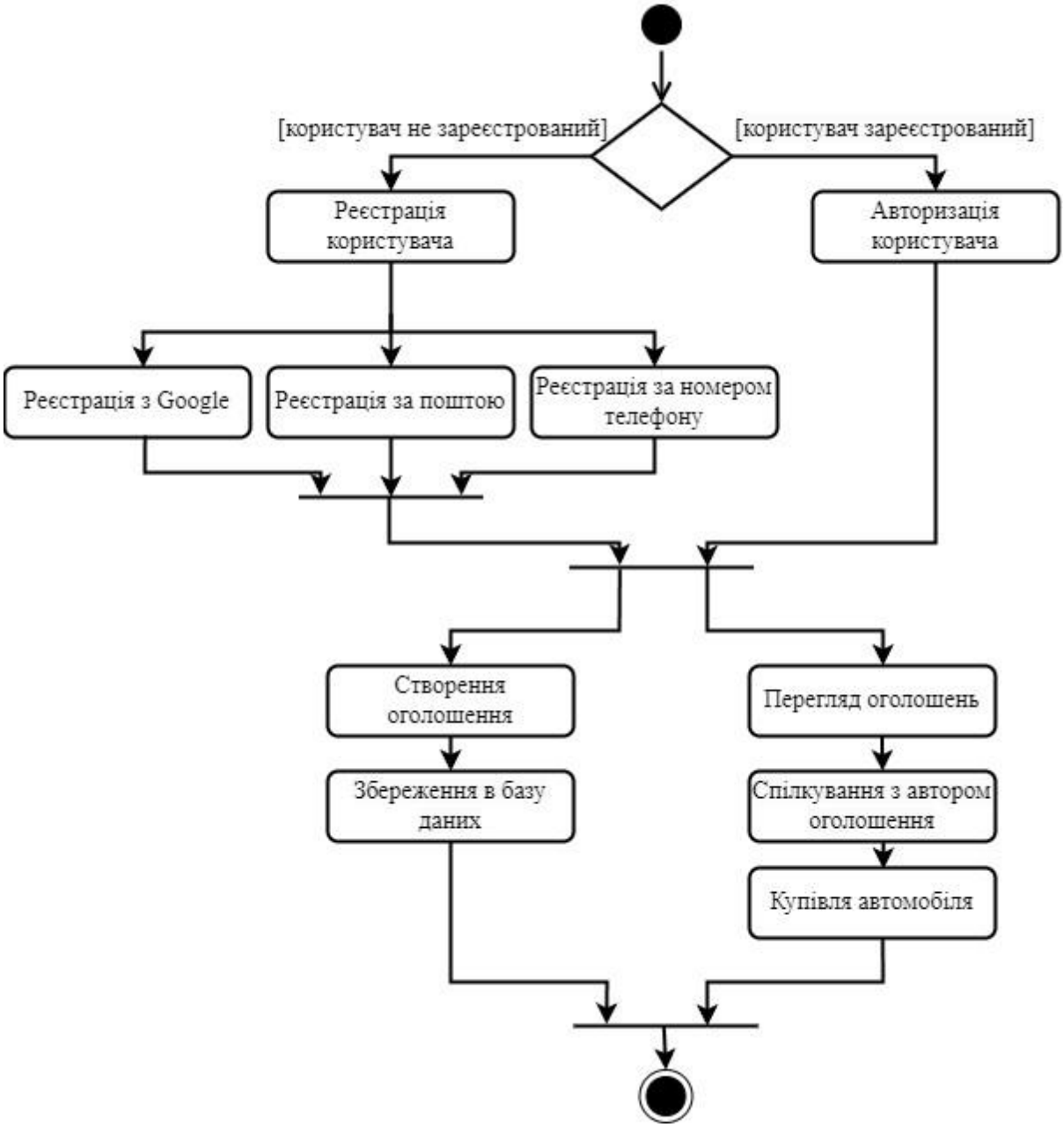

Рисунок 2.2. – Зображення UML діаграми загальної моделі програми

Діаграма забезпечує загальний огляд роботи застосунку, не занурюючись у деталі робочих процесів, щоб забезпечити простоту сприйняття та легкість розуміння.

2.4 Розробка алгоритмів роботи програми

Для розробки програмних засобів мобільної системи-консультанта для пошуку, купівлі та продажу автомобілів потрібно розробити загальний алгоритм, який визначатиме його функціональні можливості (див. рисунок 2.3).

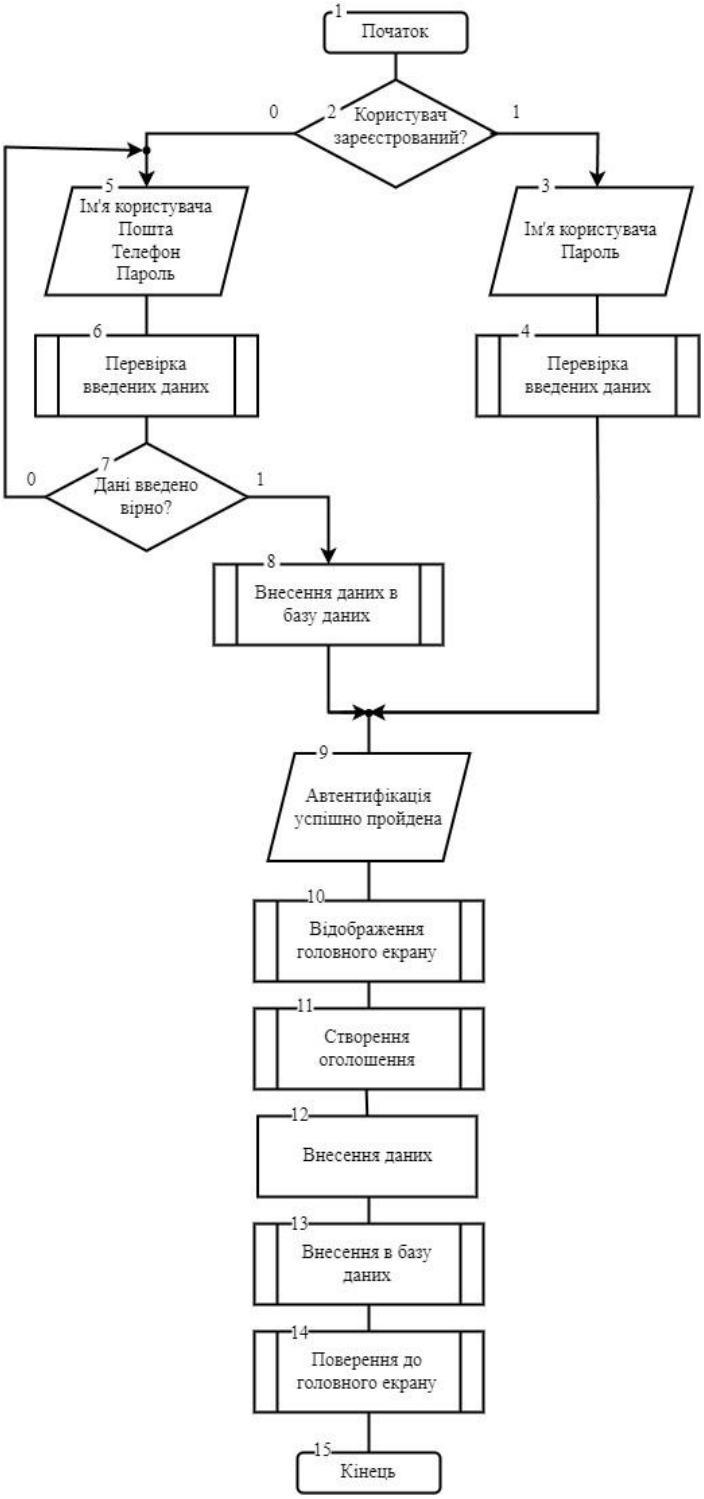

Рисунок 2.3. – Зображення загального алгоритму роботи програми

Згідно цього алгоритму користувач взаємодіє з вікном аутентифікації для входу у систему, у випадку якщо користувач не зареєстрований, йому потрібно пройти реєстрацію. Після чого дані записуються у базу даних та відкривається головний екран. На екрані мобільного пристрою користувач обирає кнопку для

створення оголошення. Наступним кроком потрібно вписати дані автомобілів у новому вікні для створення оголошень. Вказавши характеристики авто дані автоматично будуть занесені в базу даних та буде створене оголошення а користувач буде перенесений до головного екрану.

2.5 Розробка інтерфейсу програмного застосунку

Інтерфейс користувача у Android Studio створюється за допомогою XML-файлів розмітки, які описують розташування та вигляд елементів у системі, таких як кнопка, текстові поля, списки. Розмітка може бути створена вручну або за допомогою редактора розмітки в Android Studio, що дозволяє перетягувати та розміщувати елементи за допомогою графічного інтерфейсу. Крім того присутня можливість налаштовувати вигляд елементів, встановлення їх властивостей та поведінки, наприклад, обробки подій натискання на кнопку чи введення тексту в поле, за допомогою набору інструментів

Тому інтерфейс мобільного застосунку вирішено зробити максимально простим та зрозумілим для користувачів.

Загалом, інтерфейс користувача в Android Studio – це ключовий аспект розробки застосунків, який дозволяє створювати привабливі та функціональні додатки з інтуїтивно зрозумілим інтерфейсом для користувачів.

Основним кольором інтерфейсу обрано світло-сірий, оскільки даний колір додає легкості та є нейтральним. Додатковими кольорами є червоний та білий, що в свою чергу додає контрасту додатку.

При відкритті додатку, користувача зустрічатиме екран з назвою системи та логотипом (див. рисунок 2.4), який буде асоціюватися з приємними враженнями у користувачів. Такий початковий екран створює перше враження про додаток, підвищує його популярність та створює позитивний настрій у користувачів відразу ж з першого запуску програми. Це може позитивно позначитися на загальному враженні від додатку та на користувальницькому досвіді.

Рисунок 2.4. – Екран завантаження

Оскільки онлайн система передбачає реєстрацію користувачів, тому було спроектовано екран з реєстрацією (див. рисунок 2.5), на якому користувач зможе обрати один із варіантів реєстрації (див. рисунок 2.6). Якщо користувач вже зареєстрований, в такому випадку він зможе одразу перейти до екрану автентифікації (див. рисунок 2.7), де йому пропонується ввести особисті дані для входу до кабінету. Виконавши автентифікацію користувач потрапить на головний екран, де він отримає доступ до мобільного застосунку та зможе повноцінно користуватися усіма функціями.

Рисунок 2.5 – Екран вибору способу та реєстрація

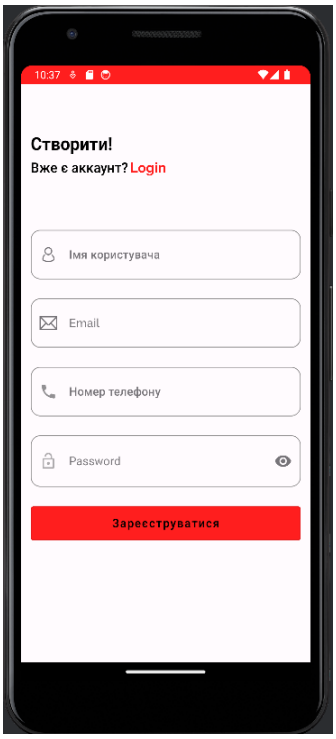

Рисунок 2.6 – Екран вибору способу та реєстрація

Якщо користувач обрав реєстрацію через електрону пошту, тоді перед ними з’явиться екран реєстрації де потрібно буде ввести дані такі як ім’я користувача, пошта, номер телефону та пароль.

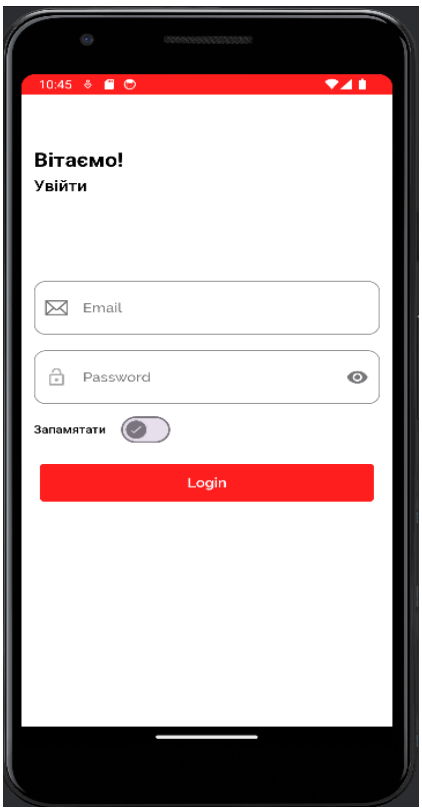

Рисунок 2.7 – Екран автентифікації користувача

На головному екрані (див. рисунок 2.8), користувач матиме доступ до пошуку, купівля та продажу автомобілів. Перейшовши в екран створення оголошень для продажу автомобілів перед користувачем з’явиться екран з вибором категорії авто (див. рисунок 2.9), після вибору категорії мобільний застосунок перейде на наступний екран в якому потрібно буде додати фотографії та всі технічні характеристики автомобіля (див. рисунок 2.10).

Після додавання оголошення, користувач може шукати автомобілі для покупки та додавати їх до вподобаних. Оголошення, які були відзначені як вподобані, будуть збережені в окремій категорії “Збережене” (див. рисунок 2.11), щоб користувач міг з легкістю знайти їх у майбутньому та повернутися до перегляду.

Цей процес дозволяє користувачам ефективно керувати своїми оголошеннями та зручно шукати автомобілі для покупки, забезпечуючи при цьому зручний та приємний досвід використання додатка.

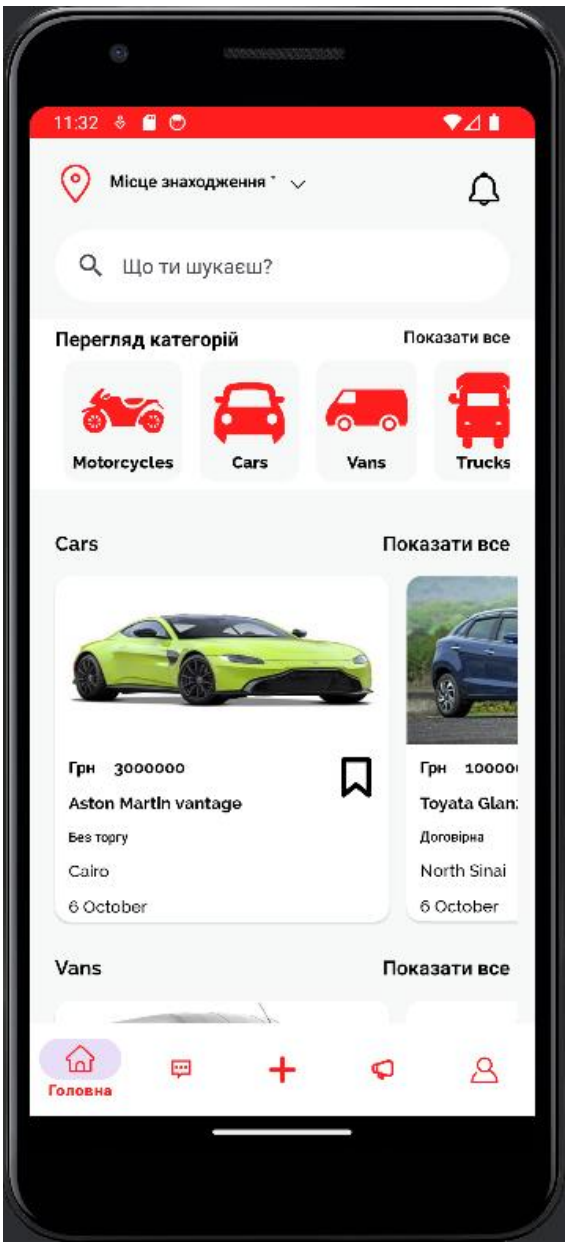

Рисунок 2.8 – Головний екран

На головному екрані користувач відразу отримає можливість переглядати оголошення автомобілів, які наявні на поточний момент. Це дозволяє користувачеві миттєво потрапити в середину великого асортименту доступних авто та ознайомитися з варіантами автомобілів. Він зможе вільно прокручувати список оголошень, вивчаючи фотографії та інформацію про кожен автомобіль безпосередньо на початковому екрані. Це забезпечує миттєвий доступ до найактуальніших пропозицій та швидкий старт у пошуку потрібного авто, що робить взаємодію з додатком максимально зручною та ефективною.

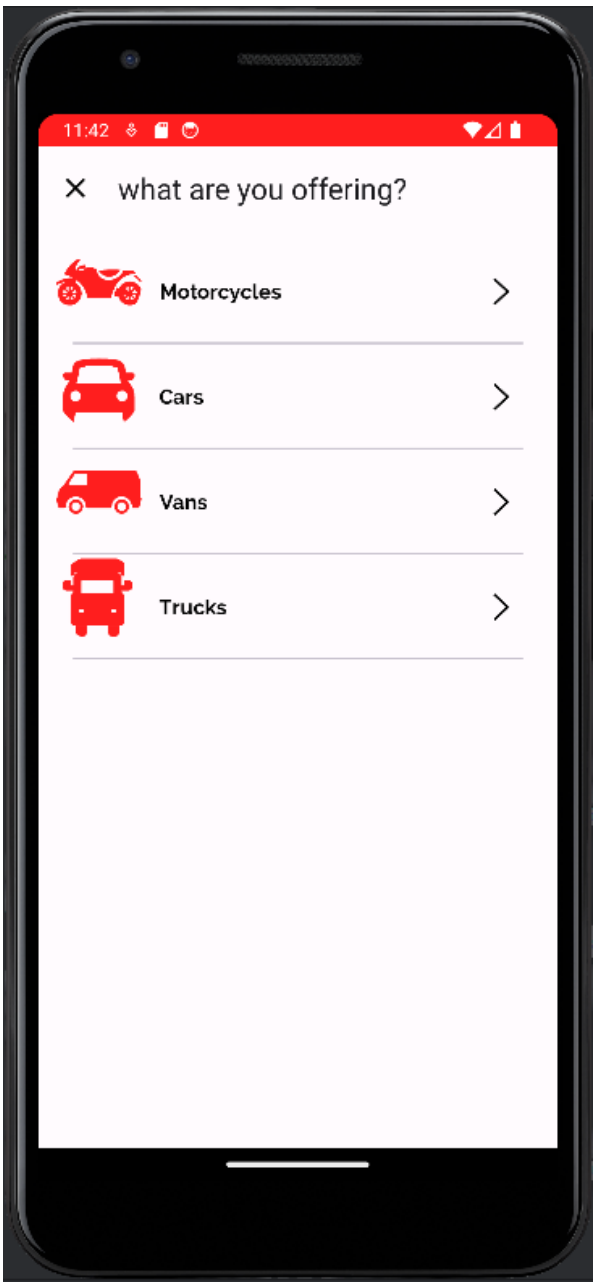

Рисунок 2.9 – Вибір категорії автомобілів

На наступному екрані користувачам потрібно буде ввести лише коректні дані про технічний стан та характеристики автомобіля. Цей крок у процесі створення оголошення є критичним, оскільки від його правильності залежить якість та достовірність інформації. Користувач повинен вводив інформацію правильно та точно, оскільки вона буде впливати на рішення потенційних покупців про купівлю автомобіля. Наприклад, неправильно вказаний рік випуску або пробіг може спричинити втрату довіри користувачів і втрату зацікавленості у покупців.

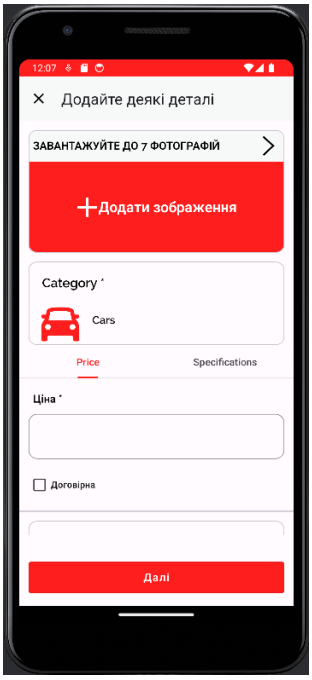

Рисунок 2.10 – Заповнення характеристик автомобілів

У особистому кабінеті користувач матиме можливість знайти всі відзначені як вподобані оголошення (див. рисунок 2.12), що дозволить зберігати та організовувати всі обрані оголошення в одному місці для подальшого перегляду та аналізу.

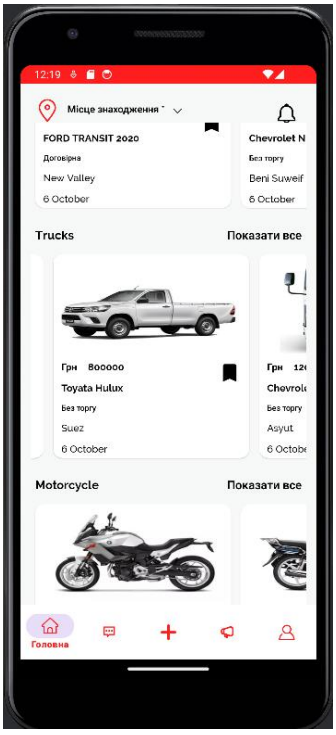

Рисунок 2.11 – Збережені оголошення автомобілів

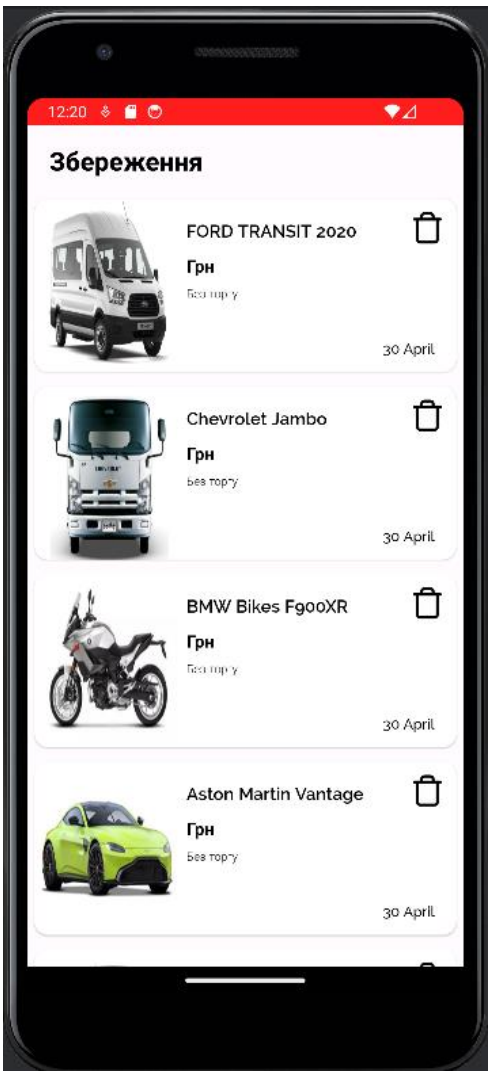

Рисунок 2.12 – Збережені оголошення автомобілів

Отже, розробка інтерфейсу користувача для мобільної системи-консультанта для пошуку, купівлі та продажу автомобілів було проведено у середовищі розробки Android Studio із використанням Kotlin.

2.6 Висновки

У розділі було проведено аналіз вхідних та вихідних даних програмного застосунку. Також було розроблено інтерфейс мобільного застосунку та загальну модель роботи мобільного застосунку за допомогою UML діаграм та загальний алгоритм роботи системи.

3 РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ КУПІВЛІ ТА ПРОДАЖУ АВТОМОБІЛІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Важним кроком у розробці мобільного застосунку для пошуку, купівлі та продажу автомобілів є вибір інструментів розробки та технологій, які будуть підтримувати та полегшувати процес. Для цієї мети було обрано використання мови програмування Kotlin з використання бібліотеки jsoup та середовища розробки Android Studio. Firebase використовується для зберігання та управління даними. Допоміжною технологією буде Jetpack Compose.

Firebase – це платформа для розробки мобільних та веб-додатків [11], яка надає різноманітні сервіси та інструменти для зручної роботи з різними аспектами додатків, такими як зберігання даних, аутентифікація користувачів, аналітика, хмарні функції. Одним з основних компонентів є Firebase Realtime Database що зберігає дані у форматі JSON та синхронізує їх в реальному часі з усіма підключеними клієнтами. Це особливо корисно для застосунків з високою динамікою оновлення даних. В Kotlin його легко інтегрувати за допомогою відповідних SDK. Firebase забезпечує комплексний підхід до розробки, дозволяючи зосередитися на функціоналі застосунку, а не на інфраструктурних питаннях. Завдяки своїй гнучкості та потужним інструментам, Firebase значно спрощує процес створення високоякісних, надійних та масштабованих застосунків.

При створенні застосунку для пошуку, купівлі та продажу автомобілів на Kotlin, бібліотека jsoup використовується для веб-парсингу даних, що може допомогти автоматично збирати інформацію про автомобілі з різних веб-сайтів. Це дозволяє інтегрувати великий обсяг даних у застосунок таких як дані про автомобілі та їх зображення , забезпечуючи користувачів актуальною інформацією.

Основні переваги використання бібліотеки jsoup:

– отримання HTML-коду веб-сторінки, що дозволяє підключитися до веб-сторінки за URL-адресою та отримати її HTML-код;

– парсинг HTML, що дозволяє аналізувати структуру та шукати потрібні елементи HTML, такі як теги або класи, щоб витягнути необхідну інформацію;

– витягування даних дозволяє достатньо легко брати текстову інформацію з вибраних елементів таку як марка, модель, ціна або рік випуску автомобілів;

– обробка та зберігання даних, що допомагає досить легко зберігати дані у базу даних або відображати їх в інтерфейсі користувача.

– автоматизація збору даних. Бібліотека jsoup дозволяє автоматизувати процес збору даних з різних веб-сайтів. А також може налаштувати регулярне сканування сайтів для отримання нових оголошень, забезпечуючи актуальність даних.

React Native – це фреймворк для розробки мобільних додатків, який дозволяє використовувати JavaScript та React для побудови нативних мобільних інтерфейсів. За допомогою React Native розробники можуть створювати застосунки для платформ iOS та Android. Він має свої переваги, включаючи швидку розробку, можливість використання одного коду для декількох платформ та доступ до великого екосистему бібліотек. Проте, він також має обмежену підтримку нативних функцій та може виникати проблеми з продуктивністю та оптимізацією, особливо в складних додатках.

Переваги React Native:

– крос-платформенність: Однією з основних переваг React Native є можливість розробляти додатки для обох платформ (Android та iOS) з використанням одного коду на JavaScript. Це дозволяє значно економити час і зусилля при розробці для обох платформ;

– швидкість розробки: React Native надає велику кількість готових компонентів та бібліотек, що дозволяє швидко створювати функціонал додатків;

– крім того, гаряче перезавантаження (hot reloading) дозволяє швидко побачити зміни у реальному часі без перекомпіляції додатку;

– платформенна інтеграція: React Native надає доступ до API платформи для інтеграції з функціоналом, таким як камера, геолокація, повідомлення тощо, що дозволяє розширити можливості додатку;

– активна спільнота та підтримка: React Native має велику та активну спільноту розробників, яка створює нові компоненти, бібліотеки та інструменти для розробки. Це сприяє швидкому вирішенню проблем та підтримці фреймворку.

Flutter – це відкрита платформа для розробки мобільних, веб-і десктопних застосунків з одного джерела коду. Він має високою продуктивність, швидкість розробки та можливість створювати красиві інтерфейси за допомогою свого власного інструментарію. Однією з ключових переваг Flutter є гаряча перезавантаження, що дозволяє швидко бачити зміни в реальному часі під час розробки. Також, оскільки Flutter ще досить молода платформа, деякі бібліотеки та рішення можуть бути менш стабільними або менше підтримуваними порівняно з іншими фреймворками.

Переваги Flutter:

– крос-платформенність: За допомогою Flutter можна створювати мобільні додатки для різних платформ, таких як Android та iOS, з використанням одного коду. Це дозволяє економити час та зусилля при розробці, оскільки не потрібно писати окремий код для кожної платформи;

– швидкодія: Flutter використовує власний движок рендерінгу Dart і не використовує мост для спілкування з нативними компонентами, що дозволяє досягти високої швидкодії та плавності відображення і анімацій;

– гарний дизайн і UI: Flutter має велику кількість готових компонентів (Widgets), які дозволяють легко створювати красивий та інтуїтивно зрозумілий інтерфейс користувача. Він також надає можливості для розробки кастомних інтерфейсів;

– гарна підтримка від Google: Flutter активно розвивається та підтримується компанією Google, що гарантує постійне оновлення і підтримку фреймворку.

Kotlin – це мова програмування, що поєднує в собі простоту використання та потужність функціональності. Вона створена компанією JetBrains і призначена для розробки різноманітних програмного забезпечення, починаючи від мобільних додатків до веб-серверів. Однією з основних причин використання Kotlin є його простота та доступність для розробників. Він має чистий та зрозумілий синтаксис Крім того, Kotlin є мовою, яка прагне забезпечити безпеку, що допомагає уникнути багатьох типових помилок. Він підтримує функціональне програмування, об'єктно-орієнтований підхід та інші сучасні парадигми програмування, що дозволяє розробникам використовувати його для будь-яких завдань.

Загалом основні преваги Kotlin включають:

– простий та зрозумілий синтаксис;
– безпека даних, що допомагає уникнути багатьох помилок;
– підтримка різних платформ та парадигм програмування.

Порівняльний аналіз мов програмування наведено в таблиці 3.1.

Таблиця 3.1. – Порівняння мов програмування

Критерій	Kotlin	Flutter	React Native
Тип	Компільована, високорівнева	Компільована, високорівнева	Інтерпретована, високорівнева
Призначення	Загальне призначення, мобільні додатки, бекенд, кросплатформенна розробка	Кросплатформенна розробка, мобільні та веб-додатки	Кросплатформенна розробка, мобільні додатки

Продовження таблиці 3.1.

Критерій	Kotlin	Flutter	React Native
Складність	Помірно складна	Легко вивчається, простий синтаксис	Легко вивчається, помірно складна
Продуктив-ність	Висока продуктивність, близька до рідних додатків	Висока продуктивність	Висока продуктивність, але може бути повільнішою за рідні додатки
Засоби тестування	Відмінні засоби тестування (JUnit, Espresso, Robolectric)	Відмінні засоби тестування (вбудовані інструменти, Mockito, Flutter Driver)	Відмінні засоби налагодження та тестування (Jest, Enzyme, Detox)
Кросплатфор менність	Висока, підтримка Android та iOS через Kotlin Multiplatform	Висока, підтримка Android, iOS, веб	Висока, підтримка Android та iOS

Jetpack Compose – це сучасна бібліотека для розробки користувацького інтерфейсу (UI) для Android-додатків [12], яка базується на мові програмування Kotlin. Вона була представлена компанією Google з метою полегшити процес розробки UI та забезпечити більш простий та ефективний спосіб створення інтерфейсів для Android-додатків.

Основні цілі використання Jetpack Compose включають:

– зручність та продуктивність Jetpack Compose надає зручний та простий спосіб опису UI за допомогою Kotlin-коду;

– широкий функціонал та інтеграція з іншими Jetpack компонентами, Jetpack Compose інтегрується з іншими компонентами Jetpack, такими як

ViewModel [13], LiveData, Room, Navigation та інші, що робить його потужним інструментом для розробки повноцінних Android-додатків;

– підтримка від Google та активна спільнота, Jetpack Compose є офіційною бібліотекою від Google, тому має активну підтримку та регулярні оновлення. Крім того, він має велику спільноту розробників, яка активно співпрацює над вдосконаленням та розвитком бібліотеки.

Також важливим є вибір середовища розробки – це програмні засоби, які забезпечують зручне середовище для всіх етапів розробки програмного забезпечення, включаючи написання коду, тестування та розгортання програм.

IntelliJ IDEA – це інтегроване середовище розробки (IDE), яке надає широкий набір інструментів для розробки програмного забезпечення на різних мовах програмування [14], таких як Java, Kotlin, Python, JavaScript та інші. Ця платформа відома своєю потужністю, продуктивністю та дружелюбним інтерфейсом, що сприяє швидкому та ефективному процесу розробки програм. Оскільки IntelliJ IDEA вимагає великих ресурсів комп'ютера, він може бути важким для використання на менш потужних комп'ютерах або для великих проектів. Також, через велику кількість функцій та складність деяких концепцій, IntelliJ IDEA може бути складним для новачків у програмуванні або для тих, хто просто починає вивчати цей інструмент.

Android Studio відзначається широкими можливостями для розробки додатків для платформи Android, включаючи різні типи пристроїв, такі як мобільні телефони, планшети та телевізори. Інтеграція з Android SDK забезпечує розробникам доступ до усіх необхідних компонентів платформи для створення потужних додатків. Android Studio має інтуїтивно зрозумілий інтерфейс користувача, що робить його привабливим для новачків у програмуванні. Підтримка різних мов програмування, таких як Java, Kotlin та C++, дає розробникам можливість вибрати мову, яка найкраще підходить для їх потреб. Крім того, великий вибір інструментів та плагінів розширює можливості Android Studio і дозволяє розробникам працювати з різноманітними інструментами для

підвищення продуктивності та якості розробки додатків. Порівняльний аналіз середовищ розробки наведено в таблиці 3.2.

Таблиця 3.2. – Порівняльний аналіз середовищ розробки

Критерії	Android Studio	IntelliJ IDEA
Підтримка Android розробки	Висока, оптимізована для Android, включає Android SDK та емулюватор	Підтримує Android розробку, але не включає Android SDK за замовчуванням
Інтеграція з Android інструментами	Вбудована інтеграція з інструментами Android	Обмежена інтеграція, потребує налаштування
Підтримка Gradle	Повна підтримка Gradle для збірки Android проектів	Підтримує Gradle, але не має специфічних для Android інструментів
Емулятор	Вбудований Android емулятора	Не має вбудованого емулятора
Оновлення та підтримка	Регулярні оновлення та підтримка від Google	Регулярні оновлення, але фокус не на Android
Інтерфейс користувача	Зручний та оптимізований для Android розробки	Зручний, але не оптимізований спеціально для Android розробки
Навчальні ресурси	Широкий вибір офіційних ресурсів	Хороша документація, але не орієнтована на Android

Отже, для розробки мобільного застосунку для пошуку, купівлі та продажу автомобілів було обрано мову програмування Kotlin та середовище розробки Android Studio, Firebase для зберігання даних та вбудовану технологію Jetpack Compose. Адже вона надає потужний та швидкий набір технологій що дозволяє швидко та зручно розробити систему-консультант для пошуку, купівлі та продажу авто.

3.2 Розробка методу представлення інформації про авторинок

Першим кроком для створення алгоритму збереження вподобаних оголошень є створення методу, який буде отримувати необхідні дані про існуючі оголошення та користувачів.

Таблиця 3.3. – Опис алгоритму збереження оголошень onViewCreated

Назва функції	Опис функції
super.onViewCreated(view, savedInstanceState)	Викликати базову реалізацію методу “onViewCreated”, для виконання збереження оголошень.
val currentUser: FirebaseUser? = firebaseAuth.currentUser val uid = currentUser?.uid	Отримати дані про поточного користувача з бази даних та його унікальний ідентифікатор.
setupSavedItemsRecyclerView() subscribeToObservables()	Викликати методи налаштування даних для відображення збережених елементів.
if (uid != null) { homeViewModel.getSavedItemsByUserId(uid, loading = true) }	Перевіти чи збережене оголошення у користувача, якщо ні, то всі дані оголошення зберігаються та переносяться до кабінету користувача.
savedItemsAdapter.setOnItemClickListener	Виконати навігацію до детальної інформації про оголошення.
savedItemsAdapter.setOnSaveItemClickListener { savedItem -> homeViewModel.removeFromSavedItemsByUserId(firebaseAuth.currentUser?.uid.toString savedItem.itemId)	Перевіти чи збережене оголошення у користувача, якщо так, то викликається метод “removeFromSavedItemsByUserId” який видаляє оголошення зі збережень поточного користувача.

CurrentUser це поточний користувач який спочатку записує поточного користувача в змінну, після чого homeViewModels перевіряє чи дані поточного користувача є в базі даних, якщо користувач існує то тоді з бази даних дістаються всі збережені елементи використовуючи метод getSavedItemsByUserId. В результаті виконання створюється кнопка яка відправляє користувача на екран з збереженими елементами за допомогою допоміжного модуля.

Таблиця 3.4. – Опис алгоритму допоміжного модуля setupSavedItemsRecyclerView

Назва функції	**Опис функції**
savedItemsRecyclerView = binding!!.savedItemsRv	Отримати дані про розмітку макета та елементів макету.
savedItemsAdapter = SavedItemAdapter()	Створити новий елемент, який відповідає за надання даних і відображення кожного елемента.
savedItemsRecyclerView.layoutManager	Розмістити елементи у вертикальному списку та відображати їх в порядку додавання.
savedItemsRecyclerView.adapter = savedItemsAdapter	Встановити адаптер “saved ItemsAdapter” щоб дані були відображені у списку.

З допомогою об’єкта SetupSavedItems починається звернення до файлу макету в якому буде відображуватися наші елементи. LayouManager створює область для відображення кожного з елементів. SavedItemsAdapter використовується для наповнення фрагменту даними про наші збережені елементи.

3.3 Розробка реєстрації та авторизації

Для початку розробки модуля реєстрації та авторизації користувачів, потрібно підключити мобільний застосунок до бази даних. Тому було створено клас FirebaseAuth. Після чого створюється змінна за допомогою якої будемо звертатися до бази даних для реєстрації.

Таблиця 3.5. – Опис модуля реєстрації користувача sendDataAndNavigate

Назва функції	**Опис функції**
firebaseAuth.currentUser	Перевірити чи зареєстрований поточний користувач за допомогою бази даних.
binding!!.inputTextLayoutUserName binding!!.inputTextLayoutPhone binding!!.inputTextLayoutEmail	Отримати дані користувача такі як ім’я, телефон, електронна пошта, у випадку якщо користувач не зареєстрований та передати до допоміжного модуля “register”.

Мобільний застосунок спочатку встановлює зв'язок із віддаленою базою даних. Під час цього процесу відбувається створення нового користувача та передача даних введених у відповідні поля за допомогою модуля register. Проте для передачі та збереження оновлених даних застосовується інший метод.

Таблиця 3.6. – Опис допоміжного методу реєстрації користувача register

Назва функції	**Опис функції**
private fun register()	Викликати базовий метод реєстрації для створення особистого кабінету користувача

Продовження таблиці 3.6.

Назва функції	Опис функції
authViewModel.registerUser binding!!.inputTextLayoutUserName, binding!!.inputTextLayoutPhone, binding!!.inputTextLayoutEmail, binding!!.inputTextLayoutPassword,	Отримати дані, введені користувачем з інтерфейсу.
authViewModel	Передати дані до бази даних для створення нового облікового запису поточного користувача.
override fun onDestroyView()	Обробити результати реєстрації.

Модуль registerUser є ключовим компонентом мобільного застосунку, який відповідає за реєстрацію нових користувачів у системі. Цей модуль виконує кілька важливих завдань, що включають в себе збір, перевірку та передачу даних користувача до віддаленої бази даних.

Щоб забезпечити функціональність авторизації користувачів у мобільному застосунку, необхідно створити клас, який взаємодіє з базою даних. Цей клас буде використовувати ініціалізатор бази даних для виконання операцій авторизації.

Для реалізації авторизації клас повинен мати метод, який перевіряє, чи існує введений користувачем логін та пароль у базі даних за допомогою LoginWithEmail. Успішна авторизація дає користувачеві доступ до функціональності застосунку, в іншому випадку виводиться повідомлення про помилку.

Таблиця 3.7. – Опис методу авторизації loginWithEmail

Назва функції	Опис функції
private fun loginWithEmail()	Викликати базовий метод авторизації

Продовження таблиці 3.7.

Назва функції	Опис функції
authViewModel.loginWithEmail(binding!!.inputTextLayoutEmail, binding!!.inputTextLayoutPassword)	Отримати дані від користувача такі як електронна пошта та пароль та передати дані до “authViewModel”
authViewModel.loginWithEmailState	Отримати дані, після чого перевірити чи існує користувач з такою електронною поштою у базі даних

Функція LoginWithEmail призначена для аутентифікації користувача за допомогою їхньої електронної адреси та паролю, які вони вводять у відповідних полях. Ця функція перевіряє, чи існує користувач з такою електронною адресою у базі даних та чи вірно введений пароль для цього облікового запису. Якщо обидва умови виконуються, користувача успішно аутентифікується.

3.4 Розробка валідації реєстрації

Перший крок валідації поля електронної пошти – перевірка наявності символа "@" у введеній стрічці. Символ "@" є обов'язковим елементом у будь-якій правильній електронній адресі, тому його наявність є ключовою для визначення правильності введеної пошти.

Таблиця 3.8. – Опис методу перевірки електронної пошти AuthValidation

Назва функції	Опис функції
fun isValidEmail(email: String): Boolean	Перевірити чи є заданий рядок, дійсною електронною адресою.

Продовження таблиці 3.8.

if (email != "")	Перевірити чи рядок з електронною поштою не порожній, якщо ні, то продовжити виконання.
val mailContainCars = !Patterns.EMAIL_ADDRESS.matcher	Перевірити електронну пошту на правильність введення.
val checkMailNumbers = email.split("@")[0] val checkIfMailNumbers: Int? = checkMailNumbers.toIntOrNull()	Перевірка довжини електронної пошти, щоб вона не була занадто короткою та занадто довгою.

Після того, як перевірено наявність символа "@", наступним кроком є перевірка довжини стрічки. Хоча точна кількість символів у частині до та після символу "@" може відрізнятися, загальним правилом є те, що електронна адреса повинна бути достатньою довгою для включення в ній імені користувача та домену. Таким чином, перевіряється, чи не є введена адреса занадто короткою.

Для перевірки пароля також використовується регулярний вираз який перевіряє чи містить введена строка не тільки цифри а й хоча б один спеціальний символ та одну літеру у великому регістрі за допомогою validatePassword.

Таблиця 3.9. – Опис методу перевірки пароля validatePassword

Назва функції	**Опис функції**
val pass = userPass.editText?.text.toString()	Отримання пароля з текстового поля від користувача.
val pat = Pattern.compile("[A-Z][^A-Z]*$") val matchCapital: Matcher = pat.matcher(pass)	Створити шаблон, який шукає рядок, що починається з великої літери і може містити будь-які символи після неї до кінця рядка.

Продовження таблиці 3.9.

val patNumber = Pattern.compile("[0-9]") val matchNumber: Matcher = patNumber.matcher(pass)	Створити шаблон, який шукає будь-яке число.
if (pass != "") {	Перевірити чи рядок пароля не пустий.
val specialPass = Pattern.compile("[.!@#\$%&*()_+=|<>?{}\\\\\\\[\\]~-]")	Створити шаблон для спеціальних символів і перевірити чи містить пароль будь-які з них. Якщо вони присутні, повідомити про недопустимість спеціальних символів у паролі.

Після успішної перевірки пароля він додається до бази даних та прив'язується до відповідного облікового запису користувача. Це важливий етап, який забезпечує користувачам доступ до їхніх облікових записів з належним захистом.

3.5 Розробка бази даних застосунку

Розробка бази даних є основою для створення ефективного мобільного застосунку персонального консультанта з пошуку, купівлі та продажу автомобілів. Вона дозволяє структурувати та організувати велику кількість інформації про автомобілі, забезпечуючи швидкий та зручний доступ до неї. Завдяки інтеграції з Firebase, дані оновлюються в режимі реального часу та зберігаються в безпечному хмарному середовищі. Використання бібліотеки Jsoup дозволяє автоматично збирати актуальну інформацію з різних джерел, гарантуючи, що користувачі отримують найсвіжіші дані про автомобілі. Крім того, база даних дає змогу аналізувати поведінку користувачів, надаючи персоналізовані рекомендації та підбираючи автомобілі, що відповідають їхнім

потребам. Захист персональних даних та фінансової інформації забезпечується за допомогою інструментів безпеки Firebase. Загалом, розробка бази даних є невід'ємною частиною створення успішного та зручного застосунку для пошуку, купівлі та продажу автомобілів.

У розроблюваному застосунку база даних використовується для керування обліковими даними користувачів, що дозволяє налаштувати автентифікацію та авторизації, також для управління життєвим циклом оголошення у застосунку.

Для кращого розуміння розроблюваної бази даних було описано діаграму класів, що зображає інформацію, яка зберігатиметься у сховищі (див. рис. 3.1).

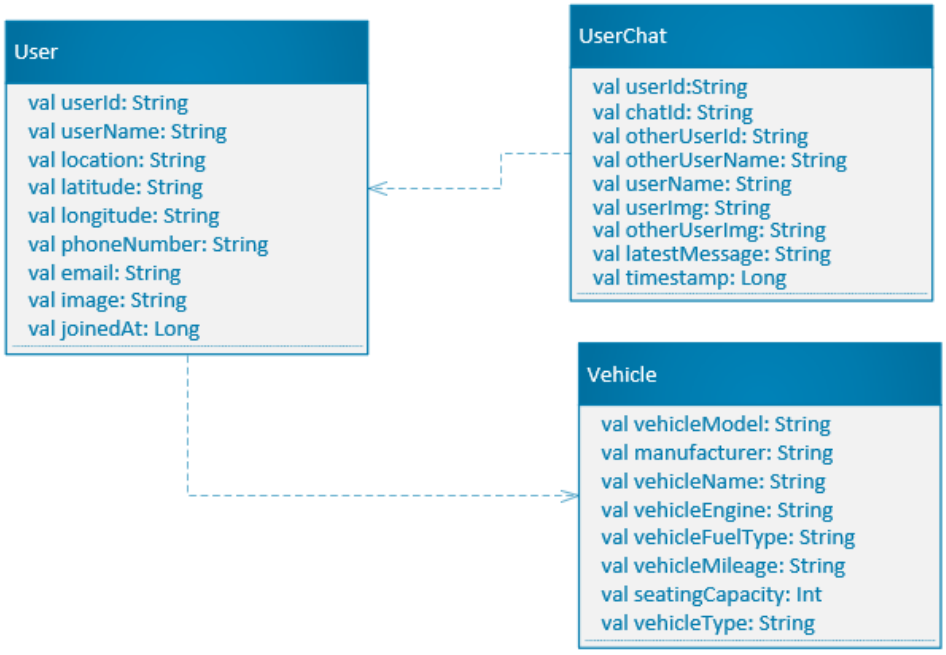

Рисунок 3.1 – Зображення діаграми основних класів

3.6 Висновки

У розділі було обґрунтовано вибір мови програмування для розробки системи та його інтерфейсу. Проаналізувавши популярні технології для розробки систем, було обрано мову програмування Kotlin та середовище розробки Android Studio. Також було описано розробку реєстрації та авторизації, та модулів валідації реєстрації та збереження вподобаних оголошень.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Аналіз методів тестування програмного забезпечення

Перевірка програмного забезпечення через тестування є ключовим етапом в процесі розробки [15], без якого неможливо забезпечити високу якість та надійність продукту. Тестування включає в себе різні підходи, методи та стратегії, спрямовані на виявлення помилок, недоліків та відхилень від вимог ще на ранніх стадіях розробки.

Цілі тестування:

– виявлення помилок і дефектів у програмі;

– перевірка відповідності програмного забезпечення встановленим вимогам;

– забезпечення стабільності та надійності продукту;

– підтвердження функціональності та продуктивності програмного забезпечення.

Ось детальніше про важливість та різновиди тестування програмного забезпечення:

– забезпечення якості тестування допомагає переконатися, що ПЗ відповідає вимогам щодо якості, функціональності та продуктивності. Використання тестування для перевірки відповідності програми до вимог допомагає впевнитися, що вона забезпечує потрібний рівень функціональності та задовольняє потреби користувачів;

– виконання вимог замовника допомагає переконатися, що програма відповідає вимогам, встановленим замовником або користувачами. Це допомагає уникнути розбіжностей та конфліктів під час подальшої експлуатації;

– підвищення впевненості користувачів полягає в якості програмного продукту, вони будуть більш схильні до його використання та рекомендацій іншим користувачам;

– виявлення помилок допомагає виявляти помилки та недоліки в програмі, які можуть призвести до некоректної роботи або навіть до аварій. Чим раніше вони виявляються, тим легше їх виправити і тим менше вони коштують;

– економія часу та коштів може здаватися витратою часу і ресурсів, але воно насправді допомагає економити час та кошти в майбутньому, уникнувши потенційних проблем та збільшивши продуктивність розробки;

– функціональне тестування це вид тестування спрямований на перевірку того [16], чи відповідає програма вимогам та функціональним специфікаціям. Включає у себе тестування різних функцій програми, вводу/виводу даних, обробки виняткових ситуацій;

– навантажувальне тестування оцінює реакцію застосунку на різні рівні навантаження. Це допомагає визначити [17], як застосунок працюватиме в умовах великої кількості одночасних користувачів або обсягу даних;

– тестування безпеки спрямоване на виявлення потенційних вразливостей та проблем з безпекою програмного забезпечення та включає у себе аналіз системи на предмет можливостей несанкціонованого доступу [18], витоку даних та інших загроз безпеці;

– тестування відновлення оцінює здатність програми до відновлення після виникнення помилок, аварій або втрати з'єднання [19]. Включає у себе тести на резервне копіювання, відновлення даних та інші сценарії відновлення.

Тестування на ранніх етапах розробки допомагає виявити помилки, що значно знижує витрати на їх виправлення, запобігає виникненню складних проблем у майбутньому і підвищує загальну якість продукту. Швидке виявлення і виправлення помилок дозволяє уникнути затримок у розробці та випуску продукту, що, в свою чергу, підвищує задоволеність кінцевих користувачів.

Під час створення мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів були розроблені відповідні тестові методи. Цей процес означає тестування окремих компонентів програмного забезпечення, таких як функції, методи, класи або модулі. Це дозволило

забезпечити ефективну роботу всієї системи та достовірність обробленої інформації. Клас для тестування валідності паролю (див. рисунок 4.1).

Рисунок 4.1 – Блок-схема алгоритму та методу тестування валідності паролю

У класі тестування валідації паролю використовується useAppContext, для перевірки, чи записаний пароль відповідно стандартам.

Тестування функціоналу є ключовою складовою в процесі розробки програмного забезпечення і орієнтується на перевірку правильності реалізації функцій застосунку згідно з вимогами та специфікаціями (див. рисунок 4.2).

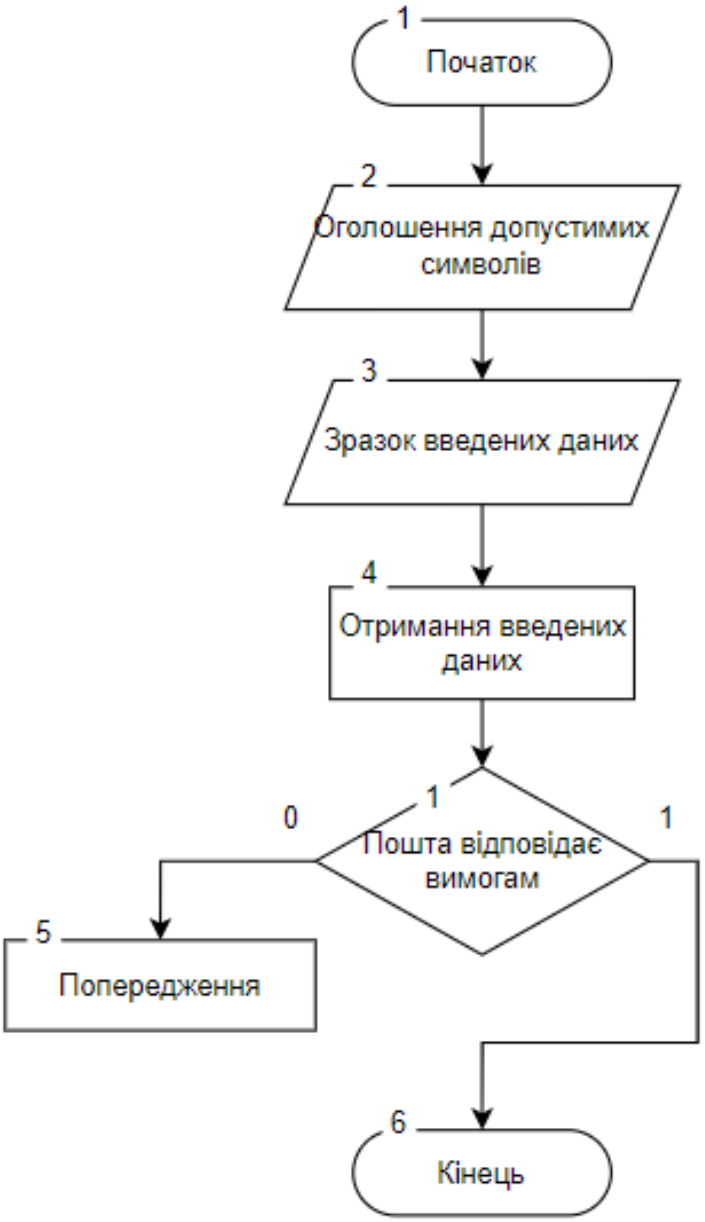

Рисунок 4.2. – Блок-схема алгоритму та методу тестування валідності електронної пошти

Клас тестування електронної пошти надає зразок варіанту введення даних та перевіряє чи дані введено вірно, якщо ні то з’являється попередження. В іншому ж випадку користувач проходить далі.

Клас тестування створення оголошень (див. рисунок 4.3), перевіряє введені дані користувачами такі як марка автомобіля, об’єм двигуна, колір, кількість пробігу тип палива. Також обов’язковим є додавання фотографій. Після чого створюється оголошення, в іншому випадку якщо дані введено не вірно або не додано зображення користувач буде повинен виправити характеристики що були введені не вірно.

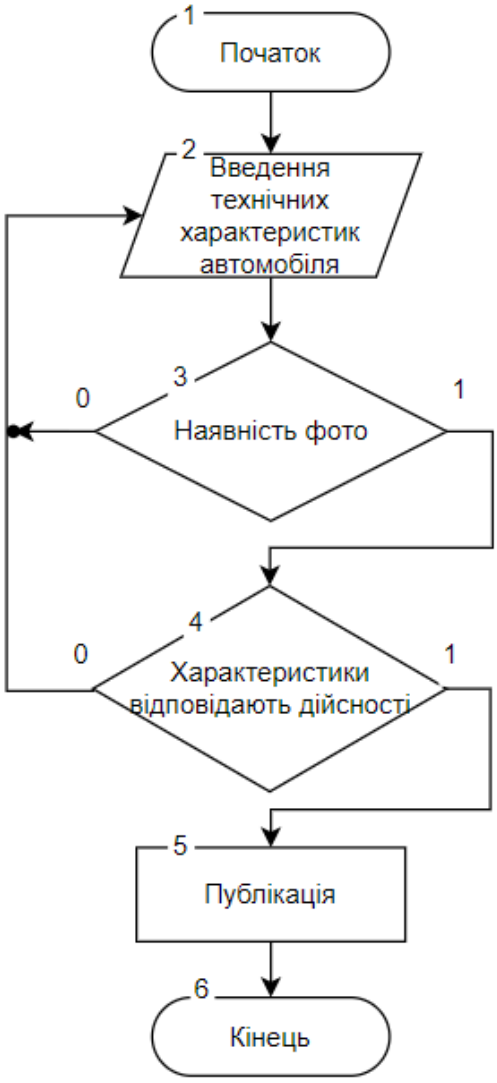

Рисунок 4.3. – Блок-схема алгоритму та методу тестування створення оголошення

Тестування функціоналу включає перевірку того, чи повертає програма очікувані результати та чи відбувається правильна обробка даних та забезпечує автоматизацію що дозволяє швидше виконувати тестування та виявлення помилок.

4.2 Розробка інструкції користувача

Інструкція користувача для мобільного застосунку – це документ, який надає повну та докладну інформацію користувачам щодо використання застосунку на їхніх мобільних пристроях [20]. Вона містить пояснення процесу встановлення та налаштування застосунку, реєстрації або входу в систему, огляд основних та додаткових функцій застосунку, інструкції щодо використання кожної функції, а також поради щодо оптимального використання застосунку. Цей документ допомагає користувачам ефективно користуватися застосунком та максимально використовувати його можливості.

Інструкція користувача становить значну складову будь-якого застосунку, оскільки вона сприяє зрозумінню користувачами того, як ефективно та безпечно використовувати застосунок.

Причини, чому розробка інструкція користувача є такою важливою:

– підтримка користувачів є важливим документом для підтримки користувачів. Вона надає відповіді на їхні запитання та допомагає вирішувати проблеми, які вони можуть зустріти;

– зменшення часу на навчання за допомогою інструкції дозволяє користувачам швидше оволодіти застосунком, що зменшує час, необхідний для навчання та адаптації до нового програмного забезпечення.

Інструкція для використання мобільного застосунку:

1. Запустити мобільний застосунок.
2. Якщо користувач не зареєстрований у системі, на початковому екрані потрібно обрати один з трьох способів реєстрації. Якщо користувач зареєстрований, то потрібно перейти до аутентифікації.

3. Пройти реєстрацію у системі, ввівши номер телефону, ім’я користувача, пошту та пароль. Натиснути на кнопку «Продовжити».

4. На початковому екрані обрати опцію «Login».

5. Ввести поля «Email» та «Password». Натиснути кнопку «Login».

6. Для початку огляду оголошень перейти на екран «Головна», натиснувши на кнопку у вигляді будинка у нижньому меню.

7. Обрати одне з оголошень та натиснути на нього.

8. В новому вікні переглянути оголошення та натиснути на кнопку «Чат з продавцем».

9. Після перегляду оголошення, у випадку якщо воно влаштовує користувача, присутня можливість збереження, для цього потрібно натиснути на кнопку у вигляді вкладки.

10. Для того щоб переглянути збережені оголошення автомобілів потрібно перейти в «Профіль» та натиснути кнопку «Збережене», після чого перед користувачем з’являться збережені дані оголошень.

11. В головному вікні обрати кнопку «Створити» для створення оголошення, що має вигляд плюсика у нижньому меню.

12. В новому вікні додати зображення, вписати всі вказані категорії характеристик та натиснути «Створити».

13. Для завершення потрібно натиснути кнопку «Вихід».

4.3 Системні вимоги

Системні вимоги – це сукупність технічних параметрів, які необхідні для встановлення та належного функціонування програмного забезпечення на комп'ютері або мобільному пристрої. Вони охоплюють як апаратні, так і програмні характеристики, які повинні бути виконані для коректної роботи програми. Системні вимоги зазвичай поділяються на мінімальні, які забезпечують базове функціонування, та рекомендовані, які гарантують оптимальну продуктивність та користувацький досвід.

Мінімальні системні вимоги зазвичай вимагають найнижчий рівень апаратного забезпечення за якими програма буде підтримувати мінімальні функції. В свою ж чергу рекомендовані системні вимоги визначають оптимальний рівень апаратного забезпечення за яким програма працювати найкращим чином.

Мінімальні системні вимоги:

– Операційна система – Android 7.0.
– Процесор – 1.5 GHz двоядерний процесор.
– Оперативна пам’ять – 2 ГБ.
– Вільне місце в смартфоні – 200 МБ.
– Розмір екрану – 720p (1280x720).
– Інтернет-з'єднання – Постійне підключення до Інтернету (Wi-Fi або мобільні дані).
– Інші вимоги – GPS модуль, будь яка камера.

Рекомендовані системні вимоги:

– Операційна система – Android 10.0.
– Процесор – 2.0 GHz чотириядерний процесор або краще.
– Оперативна пам’ять – 4 ГБ або більше.
– Вільне місце в смартфоні – 200 МБ.
– Розмір екрану – Full HD (1920x1080).
– Інтернет-з'єднання – Постійне підключення до Інтернету (Wi-Fi або мобільні дані).
– Інші вимоги – GPS модуль, Високоякісна камера для завантаження чітких фотографій автомобілів.

Ці системні вимоги забезпечать належну роботу мобільного застосунку з купівлі-продажу автомобілів, забезпечуючи користувачам зручність та швидкодію.

4.4 Висновки

У розділі було проведено тестування системи та наведено приклади розроблених тестів. Також була створена інструкція користувача для ефективного використання мобільного застосунку, який дозволить безпечно та якісно відповідати на всі питання щодо її використання на практиці.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено мобільний застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup. Розроблене програмне забезпечення має на меті покращення систем для пошуку, купівлі та продажу автомобілів.

Було проаналізовано стан мобільних застосунків пов’язаних з автомобілями, виконано порівняльний аналіз аналогів та проаналізовано напрямки удосконалення програмного застосунку для підбору автомобілів. Також було проведено аналіз методів вирішення задачі та визначено основні завдання для розробки програмного застосунку.

Було спроектовано загальну модель програми та розроблено структуру програмного застосунку. Було розроблено загальний алгоритм роботи застосунку, розроблено алгоритм реєстрації та авторизації користувачів та алгоритм для збереження вподобаних оголошень до особистого кабінету поточного користувача.

Протягом роботи над бакалаврською дипломною роботою було також розглянуто популярні мови програмування та середовища розробки, також було проведено аналіз за яким було обрано мову програмування Kotlin з використання бібліотеки jsoup та середовище розробки Android Studio

Було проведено аналіз видів тестування програмного продукту та впроваджено для забезпечення швидкої та ефективної роботи мобільного застосунку. Було визначено рекомендовані системні вимоги та розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рудковський В.В., Катєльніков Д.І.. Проблеми та перспективи використання програмних застосунків для пошуку, купівлі та продажу автомобілів. Матеріали Молодь в науці: дослідження, проблеми, перспективи (МН-2024). Вінниця 2024, [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21379.

2. OLX [Електронний ресурс] – Режим доступу до ресурсу: http://surl.li/tjiud.

3. Autoria [Електронний ресурс] – Режим доступу до ресурсу: https://1ll.ink/gxY5W.

4. Avtomoto [Електронний ресурс] – Режим доступу до ресурсу: https://www.otzyvua.net/uk/automotoua.

5. UI/UX дизайн інтерфейсу [Електронний ресурс] – Режим доступу до http://surl.li/tjivy.

6. Що таке машинне навчання [Електронний ресурс] – Режим доступу до ресурсу: https://thetransmitted.com/adlucem/shho-take-mashynne.

7. Інтеграція з популярних онлайн-платформ [Електронний ресурс] – Режим доступу до ресурсу: https://crm.ua/%D1%96ntegracz%D1%96ya-sogodn%D1%96.

8. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio.

9. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://lemon.school/blog/mova-programuvannya-kotlin.

10. What is a UML diagram [Електронний ресурс] – Режим доступу до ресурсу: https://miro.com/diagramming/what-is-a-uml-diagram/.

11. Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/.

12. Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/develop/ui/compose/tutorial.

13. ViewModel overview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/topic/libraries/architecture/viewmodel.

14. IntelliJ IDEA overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html.

15. Тестування програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/testuvannia-prohramnoho-zabezpechennia/.

16. Функціональне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/funktsionalne-testuvannya/.

17. Що таке навантажувальне тестування? [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/.

18. Тестування безпеки [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/security.

19. Тестування на відмову та поновлення [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/testuvannya-na-vidmovu-ta-ponovlennya/.

20. Інструкція користувача: User Manual [Електронний ресурс] – Режим доступу до ресурсу: https://qatestlab.com/resources/knowledge-center/sample-deliverables/user-manuals/.

ДОДАТКИ

62

Додаток А. Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
д.т.н., професор
_____ О. Н. Романюк
«12» березня 2024 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
_____ д.т.н., доцент Д. І. Катєльніков
«12» березня 2024 р.

Виконав:
_____ студент гр. 4ПІ-20б В. В. Рудковський
«12» березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup».

Галузь застосування – мобільний застосунок.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою бакалаврської дипломної роботи є процес створення та покращення мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів, що додатково буде використовувати Firebase для збереження даних користувачів та оголошень. Також використовуватиме бібліотеку jsoup для створення додаткових оголошень автомобілів.

Призначення роботи – розробка та реалізація програмного забезпечення мобільного застосунку для пошуку, купівлі та продажу автомобілів.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР:

1. Офіційна документація системи Android [Електронний ресурс] – 2018. – Режим доступу до ресурсу: https://developer.android.com.

2. Пєтух А. М. Бази даних. Мови запитів, управління транзакціями, розподілена обробка даних. Навчальний посібник [Електронний ресурс] / А. М. Пєтух, О. В. Романюк, О. Н. Романюк // ВНТУ, 2016. – Режим доступу до ресурсу: https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/.

5. Технічні вимоги

Тип програми – мобільний застосунок; модель розробки – ітеративна; засоби зберігання даних – Firebase; вхідні дані: облікові дані користувача, оголошення та інформація про транспортні засоби; вихідні дані: інформація у вигляді оголошень; середовище розробки – Android Studio; мова програмування – Kotlin; програмне забезпечення для симуляції мобільних пристроїв – Android Emulator.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз застосунку для пошуку, купівлі та продажу автомобілів та вибір цілей для поставленої задачі	13.03.24 –20.03.24
2	Розробка алгоритмів застосунку	13.03.24 –20.03.24
3	Розробка алгоритму для збереження вподобаних оголошень	22.03.24 –13.04.24
4	Розробка реєстрації та авторизації	15.04.24 –25.04.24

5	Розробка валідації реєстрації	26.04.24 –03.05.24
6	Розробка бази даних застосунку	04.05.24 –11.05.24
7	Тестування програмного забезпечення	12.05.24 – 23.05.24
8	Оформлення матеріалів до захисту БДР	25.05.24 – 03.06.24

10. Порядок контролю та прийняття

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

66

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Романюк О.Н.

Оригінальність	95.6%
Схожість	4.4%

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

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

Особа, відповідальна за перевірку Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck Рудковський В. В.

Автор роботи Катєльніков. Д. І.

Керівник роботи

Додаток В. Лістинг програми

```
//AccountItemAdapter.kt
package com.example.carzoneapp.adapters

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.AsyncListDiffer
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.carzoneapp.databinding.AccountItemBinding
import com.example.domain.entity.ProfileItem

class AccountItemAdapter : RecyclerView.Adapter<AccountItemAdapter.AccountItemViewHolder>() {

    inner class AccountItemViewHolder(val binding: AccountItemBinding) :
        RecyclerView.ViewHolder(binding.root)

    private val differCallBack = object : DiffUtil.ItemCallback<ProfileItem>() {
        override fun areItemsTheSame(oldItem: ProfileItem, newItem: ProfileItem): Boolean {
            return oldItem.id == newItem.id
        }

        override fun areContentsTheSame(oldItem: ProfileItem, newItem: ProfileItem): Boolean {
            return oldItem == newItem
        }

    }
    val differ = AsyncListDiffer(this, differCallBack)
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): AccountItemViewHolder {
        return AccountItemViewHolder(
            AccountItemBinding.inflate(
                LayoutInflater.from(parent.context), parent, false
            )
        )


    }

    override fun onBindViewHolder(holder: AccountItemViewHolder, position: Int) {
        val accountItem = differ.currentList[position]
        holder.itemView.apply {
            Glide.with(this).load(accountItem.iconOne).into(holder.binding.accountItemIcon)
            Glide.with(this).load(accountItem.iconTwo).into(holder.binding.accountItemIcon2)
            holder.binding.accountItemTitle.text = accountItem.title
            holder.binding.accountItemSubtitle.text = accountItem.subtitle
            setOnClickListener {
                onItemClickListener?.let { it(accountItem) }
            }

        }
    }

    override fun getItemCount(): Int = differ.currentList.size

    private var onItemClickListener: ((ProfileItem) -> Unit)? = null
    fun setOnItemClickListener(listener: (ProfileItem) -> Unit) {
        onItemClickListener = listener
    }
}

//AdItemAdapter.kt
package com.example.carzoneapp.adapters

import android.view.LayoutInflater
```

```
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.carzoneapp.databinding.AdImageItemBinding

class AdImageAdapter(private val imageUrls: List<String>) :
    RecyclerView.Adapter<AdImageAdapter.ImageViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ImageViewHolder {
        return ImageViewHolder(
            AdImageItemBinding.inflate(
                LayoutInflater.from(parent.context), parent, false
            )
        )
    }

    override fun onBindViewHolder(holder: ImageViewHolder, position: Int) {
        val imageUrl = imageUrls[position]
        holder.itemView.apply {
            Glide.with(this)
                .load(imageUrl)
                .into(holder.binding.adImageView)
        }

    }

    override fun getItemCount(): Int = imageUrls.size


    inner class ImageViewHolder(val binding: AdImageItemBinding) :
        RecyclerView.ViewHolder(binding.root)
}

//AdsAdapter.kt
package com.example.carzoneapp.adapters

import android.os.Build
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.annotation.RequiresApi
import androidx.recyclerview.widget.AsyncListDiffer
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.carzoneapp.R
import com.example.carzoneapp.databinding.HomeAdItemBinding
import com.example.carzoneapp.databinding.MyAdsItemBinding
import com.example.carzoneapp.helper.extractDateAndTime
import com.example.domain.entity.Ad


class AdsAdapter(
    private val type: Int,
     private val onItemClickListener: ((Ad) -> Unit)? = null,
     private val onSaveIconClickListener: ((Ad) -> Unit)? = null

) :
    RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    enum class ViewType {
        TYPE_ONE,
        TYPE_TWO
    }

    inner class AdsAdapterViewHolder(val adsBinding: HomeAdItemBinding) :
        RecyclerView.ViewHolder(adsBinding.root)

    inner class MyAdsAdapterViewHolder(val myAdsBinding: MyAdsItemBinding) :
        RecyclerView.ViewHolder(myAdsBinding.root)
```

```
    private val differCallBack = object : DiffUtil.ItemCallback<Ad>() {
        override fun areItemsTheSame(
            oldItem: Ad,
            newItem: Ad
        ): Boolean {
            return oldItem.adId == newItem.adId
        }

        override fun areContentsTheSame(
            oldItem: Ad,
            newItem: Ad
        ): Boolean {
            return oldItem == newItem
        }

    }
    val differ = AsyncListDiffer(this, differCallBack)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        return when (viewType) {
            ViewType.TYPE_ONE.ordinal -> {
                AdsAdapterViewHolder(
                    HomeAdItemBinding.inflate(
                        LayoutInflater.from(parent.context), parent, false
                    )
                )

            }

            ViewType.TYPE_TWO.ordinal -> {
                MyAdsAdapterViewHolder(
                    MyAdsItemBinding.inflate(
                        LayoutInflater.from(parent.context),
                        parent,
                        false
                    )
                )
            }

            else -> throw IllegalArgumentException("Invalid view type")
        }


    }

    @RequiresApi(Build.VERSION_CODES.O)
    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        val ads = differ.currentList[position]
        when (holder.itemViewType) {
            ViewType.TYPE_ONE.ordinal -> {
                val adsViewHolder = holder as AdsAdapter.AdsAdapterViewHolder
                adsViewHolder.itemView.apply {
                    Glide.with(this).load(ads.vehicleImages?.get(0))
                        .into(adsViewHolder.adsBinding.adImg)
                    adsViewHolder.adsBinding.adName.text = ads.adsData.title
                    adsViewHolder.adsBinding.adPrice.text = ads.adsData.price
                    adsViewHolder.adsBinding.sellerLocationTv.text= ads.adsData.location
                    if (ads.isInSavedItems == true) {

adsViewHolder.adsBinding.saveIcon.setImageResource(R.drawable.bookmark_ic_icon)
                    } else {
                        adsViewHolder.adsBinding.saveIcon.setImageResource(R.drawable.bookmark)
                    }
                    if (ads.adsData.negotiable) {
                        adsViewHolder.adsBinding.negotiableTv.text =
                            context.getString(R.string.negotiable)
                    } else {
                        adsViewHolder.adsBinding.negotiableTv.text =
                            context.getString(R.string.not_negotiable)
```

```
                    }
                    adsViewHolder.adsBinding.saveIcon.setOnClickListener {

                        onSaveIconClickListener?.invoke(ads)
                    }
                    val firebaseDate = ads.adsData.date
                    val dateAndTime = extractDateAndTime(firebaseDate!!)
                    val date = dateAndTime.day + " " + dateAndTime.month
                    adsViewHolder.adsBinding.listedDate.text = date
                    setOnClickListener {
                        onItemClickListener?.let { it(ads) }
                    }

                }
            }

            ViewType.TYPE_TWO.ordinal -> {
                val myAdsViewHolder = holder as AdsAdapter.MyAdsAdapterViewHolder
                myAdsViewHolder.itemView.apply {
                    Glide.with(this).load(ads.vehicleImages?.get(0))
                        .into(myAdsViewHolder.myAdsBinding.carImg)
                    myAdsViewHolder.myAdsBinding.carName.text = ads.adsData.title
                    myAdsViewHolder.myAdsBinding.carPrice.text = ads.adsData.price
                    myAdsViewHolder.myAdsBinding.sellerLocationTv.text= ads.adsData.location

                    if (ads.adsData.negotiable) {
                        myAdsViewHolder.myAdsBinding.negotiableTv.text =
                            context.getString(R.string.negotiable)
                    } else {
                        myAdsViewHolder.myAdsBinding.negotiableTv.text =
                            context.getString(R.string.not_negotiable)
                    }
                    val firebaseDate = ads.adsData.date
                    val dateAndTime = extractDateAndTime(firebaseDate!!)
                    val date = dateAndTime.day + " " + dateAndTime.month
                    myAdsViewHolder.myAdsBinding.listedDate.text = date
                    setOnClickListener {
                        onItemClickListener?.invoke(ads)
                    }
                }
            }
        }
    }

    override fun getItemCount(): Int = differ.currentList.size

    interface OnItemClickListener {
        fun onItemClick(childPosition: Int)
    }


    override fun getItemViewType(position: Int): Int {
        //return if (position % 2 == 0) TYPE_VIEW_HOLDER_ONE else TYPE_VIEW_HOLDER_TWO
        return when (type) {
            1 -> {
                ViewType.TYPE_ONE.ordinal
            }

            2 -> {
                ViewType.TYPE_TWO.ordinal
            }

            else -> throw IllegalArgumentException("Invalid value")

        }
    }


}
```

```
//Categorydapter.kt
package com.example.carzoneapp.adapters

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.AsyncListDiffer
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.carzoneapp.databinding.CategoryItemBinding
import com.example.carzoneapp.databinding.OfferingCategoriesLayoutBinding
import com.example.domain.entity.VehiclesCategories

class CategoryAdapter(private val type: Int) : RecyclerView.Adapter<RecyclerView.ViewHolder>() {
    enum class ViewType {
        TYPE_ONE,
        TYPE_TWO
    }
    inner class HomeViewHolder(val binding: CategoryItemBinding) :
        RecyclerView.ViewHolder(binding.root)

    inner class SellViewHolder(val sellBinding: OfferingCategoriesLayoutBinding) :
        RecyclerView.ViewHolder(sellBinding.root)

    private val differCallBack = object : DiffUtil.ItemCallback<VehiclesCategories>() {
        override fun areItemsTheSame(
            oldItem: VehiclesCategories,
            newItem: VehiclesCategories
        ): Boolean {
            return oldItem.categoryID == newItem.categoryID
        }

        override fun areContentsTheSame(
            oldItem: VehiclesCategories,
            newItem: VehiclesCategories
        ): Boolean {
            return oldItem == newItem
        }

    }
    val differ = AsyncListDiffer(this, differCallBack)
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder  {
        return when (viewType) {
            ViewType.TYPE_ONE.ordinal -> {
                HomeViewHolder(
                    CategoryItemBinding.inflate(
                        LayoutInflater.from(parent.context), parent, false
                    )
                )

            }

            ViewType.TYPE_TWO.ordinal -> {
                SellViewHolder(
                    OfferingCategoriesLayoutBinding.inflate(
                        LayoutInflater.from(parent.context),
                        parent,
                        false
                    )
                )
            }

            else -> throw IllegalArgumentException("Invalid view type")
        }


    }
```

```
    override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position: Int) {
        val category = differ.currentList[position]
        when (holder.itemViewType) {
            ViewType.TYPE_ONE.ordinal -> {
                val homeViewHolder = holder as HomeViewHolder
                homeViewHolder.itemView.apply {
                    Glide.with(this).load(category.categoryImage)
                        .into(homeViewHolder.binding.categoryImg)
                    homeViewHolder.binding.categoryName.text = category.categoryName
                    setOnClickListener {
                        onItemClickListener?.let { it(category) }
                    }
                }
            }

            ViewType.TYPE_TWO.ordinal -> {
                val sellViewHolder = holder as SellViewHolder
                sellViewHolder.itemView.apply {
                    Glide.with(this).load(category.categoryImage)
                        .into(sellViewHolder.sellBinding.sellFragmentRvImg)
                    sellViewHolder.sellBinding.sellFragmentRvTxt.text = category.categoryName
                    setOnClickListener {
                        onItemClickListener?.let { it(category) }
                    }
                }

            }
        }
    }

    override fun getItemCount(): Int = differ.currentList.size

    private var onItemClickListener: ((VehiclesCategories) -> Unit)? = null
    fun setOnItemClickListener(listener: (VehiclesCategories) -> Unit) {
        onItemClickListener = listener
    }

    override fun getItemViewType(position: Int): Int {
        //return if (position % 2 == 0) TYPE_VIEWHOLDER_ONE else TYPE_VIEWHOLDER_TWO
        return when (type) {
            1 -> {
                ViewType.TYPE_ONE.ordinal
            }

            2 -> {
                ViewType.TYPE_TWO.ordinal
            }

            else -> throw IllegalArgumentException("Invalid value")

        }
    }

}

//Chatlistadapter.kt
package com.example.carzoneapp.adapters

import android.os.Build
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.annotation.RequiresApi
import androidx.recyclerview.widget.AsyncListDiffer
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.bumptech.glide.Glide
import com.example.carzoneapp.databinding.ChatListItemBinding
import com.example.carzoneapp.helper.calculateTimeStampDifference
import com.example.domain.entity.UserChat
```

```
class ChatListAdapter(userId: String) :
RecyclerView.Adapter<ChatListAdapter.ChatListViewHolder>() {

    private val currentUserId = userId

    inner class ChatListViewHolder(val binding: ChatListItemBinding) :
        RecyclerView.ViewHolder(binding.root)

    private val differCallBack = object : DiffUtil.ItemCallback<UserChat>() {
        override fun areItemsTheSame(oldItem: UserChat, newItem: UserChat): Boolean {
            return oldItem.chatId == newItem.chatId
        }

        override fun areContentsTheSame(oldItem: UserChat, newItem: UserChat): Boolean {
            return oldItem == newItem
        }

    }
    val differ = AsyncListDiffer(this, differCallBack)
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ChatListViewHolder {
        return ChatListViewHolder(
            ChatListItemBinding.inflate(
                LayoutInflater.from(parent.context), parent, false
            )
        )


    }

    @RequiresApi(Build.VERSION_CODES.O)
    override fun onBindViewHolder(holder: ChatListViewHolder, position: Int) {
        val chatItem = differ.currentList[position]
        holder.itemView.apply {
            val currentTimestamp = System.currentTimeMillis()
            val time = calculateTimeStampDifference(chatItem.timestamp, currentTimestamp)
            val lastMessageTime= if (time.size==1){
                time[0]
            }else{
                time[0] + " " + time[1]
            }
            holder.binding.chatLastMessageTimeTv.text = lastMessageTime
            if (currentUserId == chatItem.userId) {
                Glide.with(this).load(chatItem.otherUserImg).into(holder.binding.chatUserImg)
            } else {
                Glide.with(this).load(chatItem.userImg).into(holder.binding.chatUserImg)
            }
            holder.binding.chatUsernameTv.text = chatItem.otherUserName
            holder.binding.lastMessage.text = chatItem.latestMessage
            setOnClickListener {
                onItemClickListener?.let { it(chatItem) }
            }

        }
    }

    override fun getItemCount(): Int = differ.currentList.size

    private var onItemClickListener: ((UserChat) -> Unit)? = null
    fun setOnItemClickListener(listener: (UserChat) -> Unit) {
        onItemClickListener = listener
    }
}

//Chooselocationadapter.kt
package com.example.carzoneapp.adapters

import android.view.LayoutInflater
import android.view.ViewGroup
```

```
import androidx.recyclerview.widget.AsyncListDiffer
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.example.carzoneapp.databinding.ChooseLocationItemBinding
import com.example.domain.entity.GeoName


class ChooseLocationAdapter() :
    RecyclerView.Adapter<ChooseLocationAdapter.ChooseLocationViewHolder>() {

    inner class ChooseLocationViewHolder(val binding: ChooseLocationItemBinding) :
        RecyclerView.ViewHolder(binding.root)

    private val differCallBack = object : DiffUtil.ItemCallback<GeoName>() {
        override fun areItemsTheSame(oldItem: GeoName, newItem: GeoName): Boolean {
            return oldItem.adminName1 == newItem.adminName1
        }

        override fun areContentsTheSame(oldItem: GeoName, newItem: GeoName): Boolean {
            return oldItem == newItem
        }

    }
    val differ = AsyncListDiffer(this, differCallBack)
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ChooseLocationViewHolder {
        return ChooseLocationViewHolder(
            ChooseLocationItemBinding.inflate(
                LayoutInflater.from(parent.context), parent, false
            )
        )


    }

    override fun onBindViewHolder(holder: ChooseLocationViewHolder, position: Int) {
        val geoName = differ.currentList[position]
        holder.itemView.apply {
            holder.binding.regionName.text = geoName.adminName1
            setOnClickListener {
                onItemClickListener?.let { it(geoName) }
            }

        }
    }

    override fun getItemCount(): Int = differ.currentList.size

    private var onItemClickListener: ((GeoName) -> Unit)? = null
    fun setOnItemClickListener(listener: (GeoName) -> Unit) {
        onItemClickListener = listener
    }
}

//SellDetailsfragment.kt
package com.example.carzoneapp.ui.fragments.ad

import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.activity.result.PickVisualMediaRequest
import androidx.activity.result.contract.ActivityResultContracts
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.coroutineScope
import androidx.lifecycle.repeatOnLifecycle
```

```
import androidx.navigation.fragment.findNavController
import androidx.navigation.fragment.navArgs
import com.bumptech.glide.Glide
import com.example.carzoneapp.R
import com.example.carzoneapp.adapters.TabAdapter
import com.example.carzoneapp.databinding.FragmentSellDetailsBinding
import com.example.carzoneapp.ui.viewmodel.MainViewModel
import com.example.carzoneapp.utils.EventObserver
import com.example.domain.entity.Ad
import com.example.domain.entity.CarData
import com.example.domain.entity.MotorCycleData
import com.example.domain.entity.TruckData
import com.example.domain.entity.VanData
import com.example.domain.entity.VehicleData
import com.example.domain.entity.VehiclesCategories
import com.google.android.material.tabs.TabLayoutMediator
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
import javax.inject.Inject


@AndroidEntryPoint
class SellDetailsFragment : Fragment() {
    private lateinit var tab1Fragment: PriceTabFragment
    private lateinit var tab2Fragment: SpecificationsTabFragment
    private var _binding: FragmentSellDetailsBinding? = null
    private val binding get() = _binding
    private val args: SellDetailsFragmentArgs by navArgs()
    private var category: VehiclesCategories? = null
    private val homeViewModel: MainViewModel by viewModels()
    private var uriArray: ArrayList<Uri>? = null

    @Inject
    lateinit var firebaseAuth: FirebaseAuth

    private val pickMultipleMedia =
registerForActivityResult(ActivityResultContracts.PickMultipleVisualMedia(7)) { uris ->
        uriArray = ArrayList()
        if (!uris.isNullOrEmpty()) {
            val imageUri: Uri?
            if (uris.size > 1) {
                // Multiple images selected
                for (uri in uris) {
                    uriArray?.add(uri)
                }
                imageUri = uris[0]
            } else {
                // Single image selected
                imageUri = uris[0]
                uriArray?.add(imageUri)
            }
            binding!!.addImg.setImageURI(imageUri)

        } else {
            Toast.makeText(requireContext(), "unknown error occurred", Toast.LENGTH_SHORT)
                .show()
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        // Inflate the layout for this fragment
        _binding = FragmentSellDetailsBinding.inflate(inflater, container, false)
        val adapter = TabAdapter(requireActivity())
        category = args.category
```

```
        tab1Fragment = PriceTabFragment()
        tab2Fragment = SpecificationsTabFragment.newInstance(category!!)
        adapter.addFragment(tab1Fragment, "Price")
        adapter.addFragment(tab2Fragment, "Specifications")
        binding!!.viewPager.adapter = adapter
        TabLayoutMediator(binding!!.tabLayout, binding!!.viewPager) { tab, position ->
            tab.text = adapter.getPageTitle(position)
        }.attach()
        return binding!!.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        // val args: SellDetailsFragmentArgs by navArgs()
        binding!!.categoryName.text = category!!.categoryName
        Glide.with(requireContext()).load(category!!.categoryImage).into(binding!!.categoryImg)
        subscribeToObservables()
        binding!!.addImg.setImageURI(uriArray?.get(0))
        binding!!.addImg.setOnClickListener {
            openGallery()
        }
        binding!!.nextBtn.setOnClickListener {
            uriArray?.toList()?.let { imagesList -> homeViewModel.uploadImages(imagesList) }
        }
    }


    private fun addAds(imageList: List<String>) {

        val adsData = tab1Fragment.getPriceTabData()
        val vehicle = tab2Fragment.getVehicleData()

        val vehicleType: VehicleData = when (category?.categoryName) {
            "Cars" -> CarData(tab2Fragment.getCarData())
            "Vans" -> VanData(tab2Fragment.getVanData())
            "Trucks" -> TruckData(tab2Fragment.getTruckData())
            "Motorcycles" -> MotorCycleData(tab2Fragment.getMotorcycleData())
            else -> throw IllegalArgumentException("Unknown category: ${category?.categoryName}")
        }
        val currentUser: FirebaseUser? = firebaseAuth.currentUser
        val uid = currentUser?.uid

        val ad = Ad(
            firebaseAuth.currentUser?.uid.toString() + System.currentTimeMillis(),
            adsData,
            vehicle,
            imageList,
            uid.toString(),
            vehicleType
        )

        homeViewModel.addVehicleAd(ad)
    }


    private fun openGallery() {

pickMultipleMedia.launch(PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.ImageAndV
ideo))

    }

    private fun subscribeToObservables() {
        lifecycle.coroutineScope.launch {
            lifecycle.repeatOnLifecycle(Lifecycle.State.STARTED) {
                homeViewModel.uploadImagesState.collect(
                    EventObserver(
                        onLoading = {
```

```
                        },
                        onSuccess = {
                            addAds(it)

                        },
                        onError = {
                            Toast.makeText(requireContext(), it, Toast.LENGTH_LONG)
                                .show()
                        }
                    )
                )
            }

            lifecycle.repeatOnLifecycle(Lifecycle.State.STARTED) {
                homeViewModel.addVehicleAdsState.collect(
                    EventObserver(
                        onLoading = {},
                        onSuccess = {

findNavController().navigate(R.id.action_sellDetailsFragment_to_homeFragment)
                            Toast.makeText(requireContext(), it, Toast.LENGTH_LONG).show()
                        },
                        onError = {
                            Toast.makeText(requireContext(), it, Toast.LENGTH_LONG).show()
                        }
                    )
                )
            }

        }
    }

}

//Setupfragment.kt
package com.example.carzoneapp.ui.fragments.auth

import android.Manifest
import android.annotation.SuppressLint
import android.content.ContentValues.TAG
import android.content.IntentSender
import android.location.Geocoder
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.text.Editable
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.activity.result.PickVisualMediaRequest
import androidx.activity.result.contract.ActivityResultContracts
import androidx.annotation.RequiresApi
import androidx.core.view.isVisible
import androidx.core.widget.doAfterTextChanged
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.coroutineScope
import androidx.lifecycle.repeatOnLifecycle
import androidx.navigation.NavOptions
import androidx.navigation.fragment.findNavController
import androidx.navigation.fragment.navArgs
import com.bumptech.glide.Glide
import com.example.carzoneapp.R
import com.example.carzoneapp.databinding.FragmentSetupBinding
import com.example.carzoneapp.helper.convertLatLongToLocation
import com.example.carzoneapp.ui.viewmodel.AuthViewModel
```

```
import com.example.carzoneapp.ui.viewmodel.MainViewModel
import com.example.carzoneapp.utils.Constants
import com.example.carzoneapp.utils.EventObserver
import com.example.domain.entity.LaunchInfo
import com.example.domain.entity.User
import com.google.android.gms.common.api.ResolvableApiException
import com.google.android.gms.location.FusedLocationProviderClient
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationServices
import com.google.android.gms.location.LocationSettingsRequest
import com.google.android.gms.location.LocationSettingsResponse
import com.google.android.gms.location.Priority
import com.google.android.gms.location.SettingsClient
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.tasks.Task
import com.google.firebase.auth.FirebaseAuth
import com.vmadalin.easypermissions.EasyPermissions
import com.vmadalin.easypermissions.dialogs.SettingsDialog
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
import timber.log.Timber
import java.io.IOException
import java.util.Locale
import javax.inject.Inject

@AndroidEntryPoint
class SetupFragment : Fragment(), EasyPermissions.PermissionCallbacks {

    private var _binding: FragmentSetupBinding? = null
    private val binding get() = _binding
    private lateinit var fusedLocationClient: FusedLocationProviderClient
    private lateinit var latLng: LatLng
    private lateinit var geocoder: Geocoder
    private val authViewModel: AuthViewModel by viewModels()
    private var user: User? = null
    private val args: SetupFragmentArgs by navArgs()
    private var uriArray: ArrayList<Uri>? = null
    private val homeViewModel: MainViewModel by viewModels()
    private var isNotLogin = false
    private var isLoginWithGoogle = false

    @Inject
    lateinit var firebaseAuth: FirebaseAuth
    private val navOptions =
        NavOptions.Builder()
            .setPopUpTo(R.id.setupFragment, true)
            .build()
    private val pickMedia =
        registerForActivityResult(ActivityResultContracts.PickVisualMedia()) { uri ->
            if (uri != null) {
                uriArray = ArrayList()
                binding!!.profileImg.setImageURI(uri)
                uriArray?.add(uri)
            } else {
                Toast.makeText(requireContext(), "Unknown error occurred", Toast.LENGTH_SHORT)
                    .show()
            }
        }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        // Inflate the layout for this fragment
        _binding = FragmentSetupBinding.inflate(inflater, container, false)
        return binding!!.root
    }

    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
```

```
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding!!.inputTextEmail.doAfterTextChanged {
            binding!!.inputTextLayoutEmail.isHelperTextEnabled = false
        }
        binding!!.inputTextPhone.doAfterTextChanged {
            binding!!.inputTextLayoutPhone.isHelperTextEnabled = false
        }
        binding!!.inputTextUserName.doAfterTextChanged {
            binding!!.inputTextLayoutUserName.isHelperTextEnabled = false
        }
        user = args.userData
        displayUserData(user!!)

        geocoder = Geocoder(requireContext(), Locale.getDefault())
        checkLocationSettings()
        subscribeToObservables()
        binding!!.locationCard.setOnClickListener {
            checkLocationSettings()
        }

        binding!!.profileImg.setOnClickListener {
            openGallery()
        }
        binding!!.LetsGoBtn.setOnClickListener {

            if (isLoginWithGoogle) {
                saveUserData(user!!.image)
            }
            uriArray?.let { image -> homeViewModel.uploadImages(image) }
        }
    }

    private fun openGallery() {

pickMedia.launch(PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.ImageOnly))
    }

    private fun displayUserData(user: User) {
        if (user.userName.isNotEmpty()) {
            binding!!.inputTextLayoutUserName.editText?.text =
                Editable.Factory.getInstance().newEditable(user.userName)
            binding!!.inputTextLayoutUserName.editText?.isEnabled = false
        }

        if (user.phoneNumber.isNotEmpty()) {
            binding!!.inputTextLayoutPhone.editText?.text =
                Editable.Factory.getInstance().newEditable(user.phoneNumber)
            binding!!.inputTextLayoutPhone.editText?.isEnabled = false
        }
        if (user.email.isNotEmpty()&&user.phoneNumber.isNotEmpty()){
            isNotLogin=true
        }
        if (user.email.isNotEmpty()) {
            binding!!.inputTextLayoutEmail.editText?.text =
                Editable.Factory.getInstance().newEditable(user.email)
            binding!!.inputTextLayoutEmail.editText?.isEnabled = false
        }
        if (user.image.isNotEmpty()) {
            Glide.with(requireContext()).load(user.image).into(binding!!.profileImg)
            binding!!.profileImg.isClickable = false
            isLoginWithGoogle = true
        }

    }

    private fun subscribeToObservables() {
        lifecycle.coroutineScope.launch {
            lifecycle.repeatOnLifecycle(Lifecycle.State.STARTED) {
```

```
                homeViewModel.uploadImagesState.collect(
                    EventObserver(
                        onLoading = {
                            binding!!.spinKitProgress.isVisible = true
                        },
                        onSuccess = { img ->
                            binding!!.spinKitProgress.isVisible = false
                            saveUserData(img[0])
                        },
                        onError = {
                            binding!!.spinKitProgress.isVisible = false
                            Toast.makeText(requireContext(), it, Toast.LENGTH_LONG)
                                .show()
                        }
                    )
                )
            }
        }
        lifecycle.coroutineScope.launch {
            lifecycle.repeatOnLifecycle(Lifecycle.State.STARTED) {
                authViewModel.saveUserDataState.collect(
                    EventObserver(
                        onLoading = {
                            binding!!.spinKitProgress.isVisible = true

                        },
                        onSuccess = {
                            binding!!.spinKitProgress.isVisible = false
                            if (isNotLogin) {

findNavController().navigate(R.id.action_setupFragment_to_loginFragment)
                            } else {
                                homeViewModel.saveFirstTimeLaunch(
                                    LaunchInfo(
                                        isFirstTimeLaunch = true,
                                        isLogin = true
                                    )
                                )
                            }
                        },
                        onError = {
                            binding!!.spinKitProgress.isVisible = false
                            Toast.makeText(requireContext(), it, Toast.LENGTH_SHORT).show()
                        }
                    )
                )
            }

        }
        lifecycle.coroutineScope.launch {
            lifecycle.repeatOnLifecycle(Lifecycle.State.STARTED) {
                homeViewModel.saveFirstTimeLaunchState.collect(
                    EventObserver(
                        onLoading = {
                            binding!!.spinKitProgress.isVisible = true
                        },
                        onSuccess = {
                            binding!!.spinKitProgress.isVisible = false
                            findNavController().navigate(R.id.homeFragment, null, navOptions)
                          //
findNavController().navigate(R.id.action_setupFragment_to_homeFragment)
                        },
                        onError = {
                            binding!!.spinKitProgress.isVisible = false

                            Toast.makeText(requireContext(), it, Toast.LENGTH_SHORT).show()
                        }
                    )
                )
```

```
            }

        }

    }

    private fun saveUserData(image: String) {
        val userId = firebaseAuth.currentUser?.uid
        val name = binding!!.inputTextLayoutUserName.editText?.text.toString()
        val email = binding!!.inputTextLayoutEmail.editText?.text.toString()
        val phone = binding!!.inputTextLayoutPhone.editText?.text.toString()
        val location = convertLatLongToLocation(
            latLng.latitude,
            latLng.longitude,
            geocoder
        )
        val user = User(
            userId!!,
            name,
            location,
            latLng.latitude.toString(),
            latLng.longitude.toString(),
            phone,
            email,
            image,
            System.currentTimeMillis()
        )
        authViewModel.saveUserData(user)
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    private val locationPermission = arrayOf(
        Manifest.permission.ACCESS_FINE_LOCATION,
        Manifest.permission.ACCESS_COARSE_LOCATION,
        Manifest.permission.ACCESS_WIFI_STATE
    )

    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    private val locationRequestLauncher =
        registerForActivityResult(ActivityResultContracts.RequestMultiplePermissions()) {
permissions ->
            val grantedPermissions = permissions.entries.filter { it.value }.map { it.key
}.toList()
            val deniedPermissions = permissions.entries.filter { !it.value }.map { it.key
}.toList()
            if (grantedPermissions.size == locationPermission.size) {
                // All permissions are granted, proceed with the desired operation
                checkLocationSettings()
            } else {
                // Some permissions are denied, handle accordingly (e.g., show a message, disable
functionality)
                if (EasyPermissions.somePermissionPermanentlyDenied(this, deniedPermissions)) {
                    // Some permissions are permanently denied, show a dialog to open app
settings
                    SettingsDialog.Builder(requireContext()).build().show()
                } else {
                    // Request the denied permissions again
                    requestLocationPermission()
                }
            }
        }

    private fun hasLocationPermission(): Boolean {
        return EasyPermissions.hasPermissions(
            requireContext(),
```

```
            Manifest.permission.ACCESS_FINE_LOCATION,
            Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_WIFI_STATE
        )
    }

    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    private fun requestLocationPermission() {
        locationRequestLauncher.launch(locationPermission)
    }

    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    @SuppressLint("MissingPermission")
    private fun getLocation() {
        fusedLocationClient = LocationServices.getFusedLocationProviderClient(requireActivity())
        fusedLocationClient.lastLocation.addOnSuccessListener { location ->
            if (location != null) {
                latLng = LatLng(location.latitude, location.longitude)
                showLocation()
            } else {
                // Location is null
            }
        }.addOnFailureListener { exception ->
            // Location request failed
            Timber.tag("fusedLocationClientException").d(exception)
        }
    }

    @SuppressLint("SetTextI18n")
    private fun showLocation() {
        try {
            val location = convertLatLongToLocation(latLng.latitude, latLng.longitude, geocoder)
            binding?.locationTv?.text = location
        } catch (e: IOException) {
            Timber.tag(TAG).e("Geocoder IOException: %s", e.message)
            binding?.locationTv?.text = "Location data not available"
        }
    }


    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    private fun checkLocationSettings() {
        if (hasLocationPermission()) {
            val locationRequest =
                LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY, 1000)
            val builder =
                LocationSettingsRequest.Builder().addLocationRequest(locationRequest.build())
            val client: SettingsClient = LocationServices.getSettingsClient(requireActivity())
            val task: Task<LocationSettingsResponse> =
client.checkLocationSettings(builder.build())
            task.addOnSuccessListener {
                // Location settings are satisfied, get location
                getLocation()
            }
            task.addOnFailureListener { exception ->
                if (exception is ResolvableApiException) {
                    // Location settings are not satisfied, show dialog to enable location
                    try {
                        exception.startResolutionForResult(
                            requireActivity(), Constants.REQUEST_CHECK_SETTINGS
                        )
                    } catch (sendEx: IntentSender.SendIntentException) {
                        //Ignore the error
                    }
                }
            }
        } else {
            requestLocationPermission()
        }
```

```
    }

    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    override fun onPermissionsGranted(requestCode: Int, perms: List<String>) {
        // Permissions granted, proceed with the desired operation
        checkLocationSettings()
    }

    @RequiresApi(Build.VERSION_CODES.TIRAMISU)
    override fun onPermissionsDenied(requestCode: Int, perms: List<String>) {
        // Permissions denied, handle accordingly (e.g., show a message, disable functionality)
        if (EasyPermissions.somePermissionPermanentlyDenied(this, perms)) {
            // Some permissions are permanently denied, show a dialog to open app settings
            SettingsDialog.Builder(requireContext()).build().show()
        } else {
            // Request the denied permissions again
            requestLocationPermission()
        }
    }
}
```

Додаток Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ПЕРСОНАЛЬНОГО КОНСУЛЬТАНТА ДЛЯ ПОШУКУ, КУПІВЛІ ТА ПРОДАЖУ АВТОМОБІЛІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ KOTLIN, FIREBASE ТА БІБЛІОТЕКИ JSOUP

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Бакалаврська дипломна робота на тему:
«Розробка мобільного застосунку персонального консультанта для пошуку, купівлі та продажу автомобілів з використанням технологій Kotlin, Firebase та бібліотеки jsoup»

Автор: ст. гр. 4ПІ-20б Рудковський В.В.
керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.

Вінниця - 2024

Рисунок Г.1 – Титульний слайд

Актуальність теми дослідження

- **Зручність та ефективність**. Завдяки мобільним застосункам, пошук та купівля автомобілів стає ефективним та приємним процесом для широкого кола користувачі.

- **Обмеженість функціоналу.** Досить багато аналогів мають обмеження в можливостях, наприклад, вони можуть не мати важливих функцій, таких як можливість порівняння автомобілів або отримання історії обслуговування та ДТП.

Рисунок Г.1 – Актуальність теми дослідження

Мета, об’єкт та предмет дослідження

Метою роботи є підвищення ефективності операцій пошуку, купівлі та продажу автомобіля шляхом розробки мобільного застосунку персонального консультанта, який дозволить оперативно оновлювати інформацію, фільтрувати, надавати рекомендації та підтримувати комунікацію з контрагентами.

Об’єктом дослідження є процес пошуку, купівлі та продажу автомобілів за допомогою мобільних застосунків.

Предметом дослідження є засоби реалізації мобільної застосунку персонального консультанта, орієнтованого на покращення досвіду корситувача у купівлі та продажу авто.

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

Задачі дослідження

- розробити графічний інтерфейс;
- розробити модуль розміщення оголошень;
- розробити модуль авторизації та реєстрації користувачів;
- розробити модуль збереження вподобаних оголошень;
- розробити модуль валідації реєстрації;
- розробити метод збереження зареєстрованих користувачів до бази даних Firebase;
- розробити систему зв’язку між покупцями та продавцями;
- провести тестування мобільного застосунку персонального консультанта.

Рисунок Г.4 – Задачі дослідження

Новизна і практична цінність отриманих результатів

Новизна. Подальшого розвитку отримав метод представлення інформації про авторинок, у якому на відміну від існуючих методів представлення інформації, комбінується інформація про наявні на поточний момент авто, яка оперативно оновлюється, та параметри вподобаних авто, які накопичуються з часом, дозволяючи користувачу формувати авто своєї мрії.

Практична цінність. Практична цінність полягає у розробці мобільного застосунку персонального консультанта, який можливо використовувати для пошуку, купівлі та продажу автомобілей.

Рисунок Г.5 – Новизна і практична цінність отриманих результатів

Порівняльний аналіз аналогів

Система	Переваги	Недоліки
OLX	Великий вибір, зручний пошук, можливість спілкуватися з продавцями безпосередньо	Багато шахраїв, не завжди є можливість перевірити історію автомобіля
Auto.ria	Великий вибір, зручний пошук, можливість перевірити історію автомобіля	Платний доступ до деяких функцій
Avtomoto.ua	Широка база даних автомобілів, можливість порівняти ціни, допомога з оформленням документів	Не такий великий вибір, як на OLX та Auto.ria
Нова система	Об'єднання всіх функцій, персоналізація, інновації, конкурентоспроможні ціни	Імовірність шахрайства

Рисунок Г.6 – Порівняльний аналіз аналогів

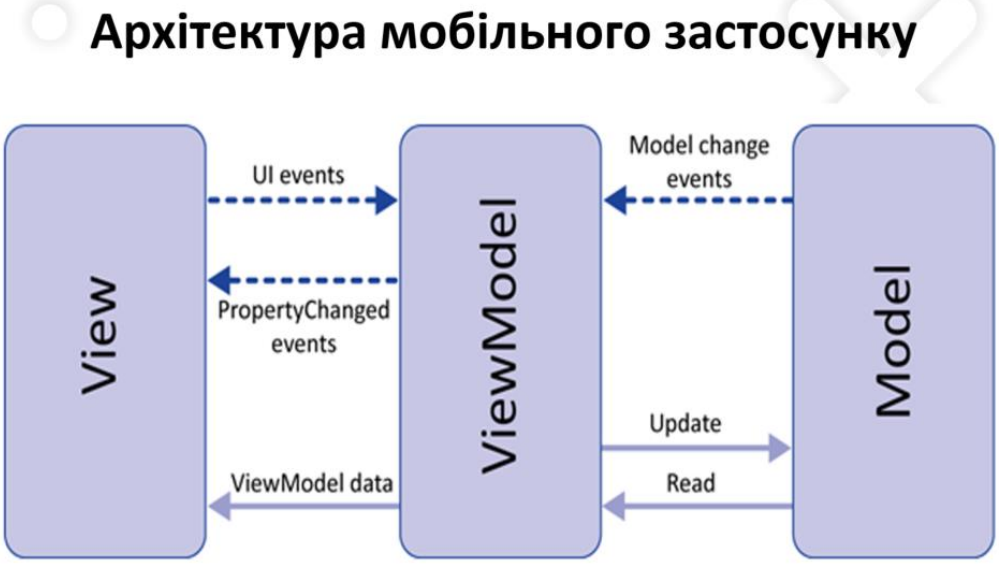

Рисунок Г.7 – Архітектура мобільного застосунку

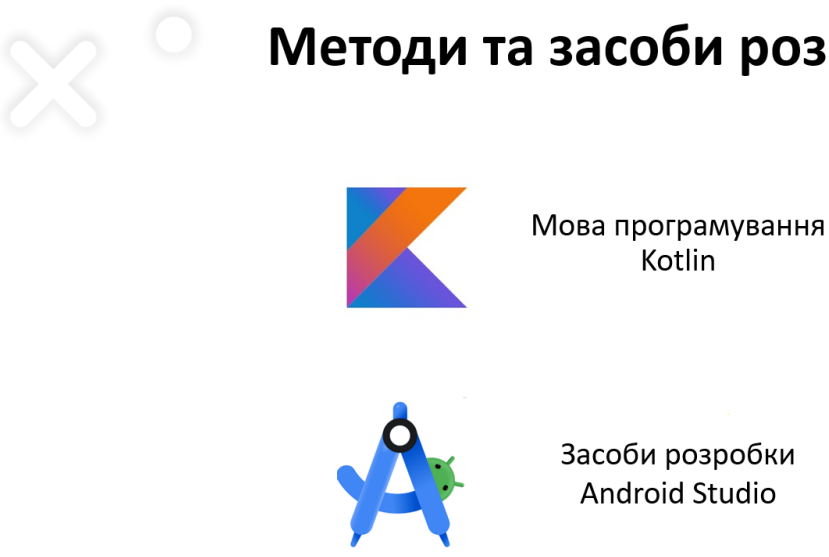

Рисунок Г.8 – Методи та засоби дослідження

Загальний алгоритм роботи системи

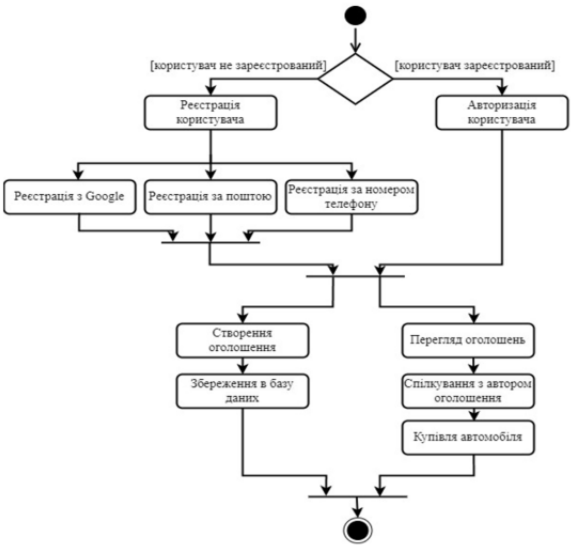

Рисунок Г.9 – Загальний алгоритм роботи системи

Розробка алгоритму для збереження вподобаних оголошень

Збереження оголошень:

1. Користувач відзначає оголошення як вподобане.
2. Перевірка користувача в базі даних.
3. Перевірка чи збережене оголошення у користувача.
4. Обробка даних про оголошення.
5. Отримання даних про розмітку макету та елементів макету.
6. Розміщення елементів у вертикальному списку.
7. Збереження оголошення до особистого кабінету користувача.

Видалення оголошень:

1. Користувач повторно відзначає оголошення як вподобане.
2. Перевірка чи збережене оголошення у поточного користувача.
3. Видалення оголошення.

Назва функції	Опис функції
super.onViewCreated(view, savedInstanceState)	Викликати базову реалізацію методу "onViewCreated", для виконання збереження оголошень.
val currentUser: FirebaseUser? = firebaseAuth.currentUser val uid = currentUser?.uid	Отримати дані про поточного користувача з бази даних та його унікальний ідентифікатор.
setupSavedItemsRecyclerView() subscribeToObservables()	Викликати методи налаштування даних для відображення збережених елементів.
if (uid != null) { homeViewModel.getSavedItemsByUserId(uid, loading = true) }	Перевіти чи збережене оголошення у користувача, якщо ні, то всі дані оголошення зберігаються та переносяться до кабінету користувача.
savedItemsAdapter.setOnItemClickListener	Виконати навігацію до детальної інформації про оголошення.
savedItemsAdapter.setOnSaveItemClickListener { savedItem -> homeViewModel.removeFromSavedItemsByUserId(firebaseAuth.currentUser?.uid.toString savedItem.itemId)	Перевіти чи збережене оголошення у користувача, якщо так, то викликається метод "removeFromSavedItemsByUserId" який видаляє оголошення зі збережень поточного користувача.

Рисунок Г.10 – Розробка алгоритму для збереження вподобаних оголошень

Демонстрація інтерфейсу застосунку «Початковий екран»

Рисунок Г.11 – Демонстрація інтерфейсу застосунку «Початковий екран»

Демонстрація інтерфейсу застосунку «Екрани реєстрації та авторизації»

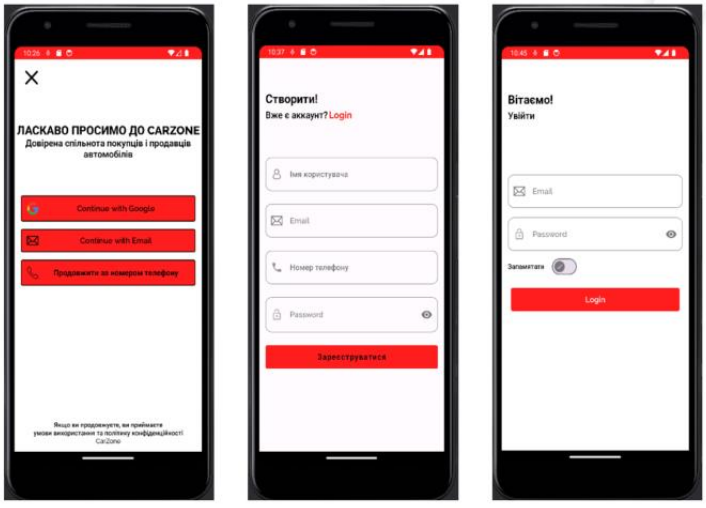

Рисунок Г.12 – Демонстрація інтерфейсу застосунку «Екрани реєстрації та авторизації»

Демонстрація інтерфейсу застосунку «Головний екран»

Рисунок Г.13 – Демонстрація інтерфейсу застосунку «Головний екран»

Демонстрація інтерфейсу застосунку «Створення оголошення»

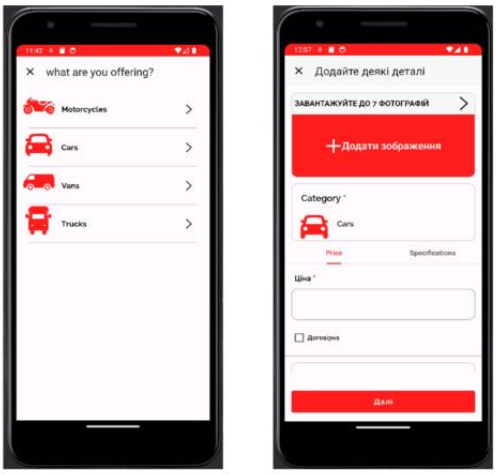

Рисунок Г.14 – Демонстрація інтерфейсу застосунку «Створення оголошення»

Демонстрація інтерфейсу застосунку «Збереження оголошень»

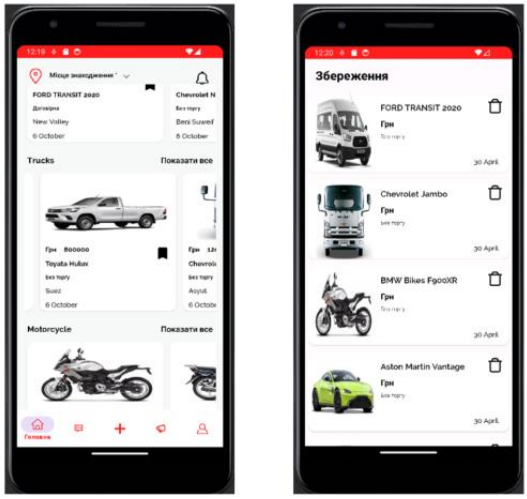

Рисунок Г.15 – Демонстрація інтерфейсу застосунку «Збереження оголошень»

Тестування валідності паролю та електронної пошти

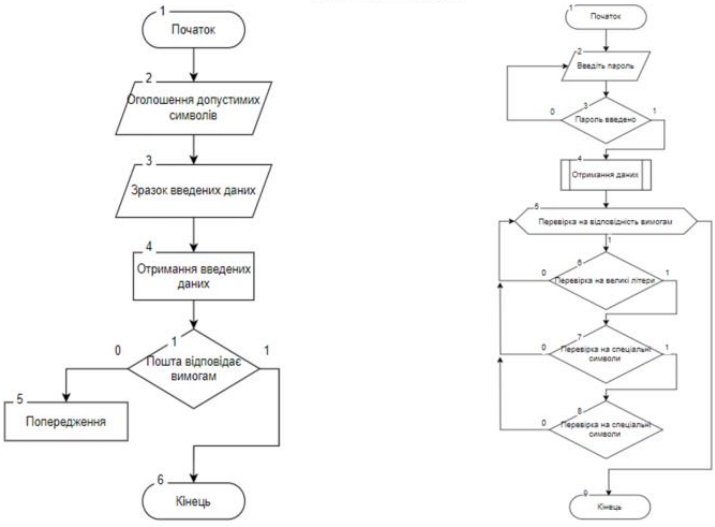

Рисунок Г.16 – Тестування валідності паролю та електронної пошти

Тестування алгоритму створення оголошень

Рисунок Г.17 – Тестування алгоритму створення оголошень

Висновки

• Розглянуто стан мобільних застосунків пов’язаних з автомобілями та методи розв’язання задачі, виконано порівняльний аналіз аналогів.
• Розроблено загальний алгоритм роботи застосунку.
• Удосконалено метод представлення інформації про авторинок, у якому на відміну від існуючих методів представлення інформації, комбінується інформація про наявні на поточний момент авто, яка оперативно оновлюється, та параметри вподобаних авто, які накопичуються з часом, дозволяючи користувачу формувати авто своєї мрії.
• Розглянуто переваги й недоліки методів програмування та засобів розробки.
• Проведено аналіз видів тестування та розроблено тестування програмної системи.
• Розроблено інструкцію користувача.

Рисунок Г.18 – Висновки

Апробація та публікація матеріалів бакалаврської дипломної роботи

Результати роботи доповідалися на:

- Молодь в науці: дослідження, проблеми, перспективи (МН-2024), Вінниця 2024.

За тематикою дослідження опубліковано одну наукову працю у збірниках матеріалів конференцій.

Рисунок Г.19 – Апробація та публікація матеріалів бакалаврської дипломної роботи

Дякую за увагу!

Рисунок Г.20 – Фінальний слайд