Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

_____Інститут електроенергетики_____
(інститут)
_____факультет інформаційних технологій_____
(факультет)
Кафедра__інформаційних технологій та комп'ютерної інженерії___
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня ___магістра__
(бакалавра, спеціаліста, магістра)

Студента _____Волошина Дмитра Андрійовича_____
(ПІБ)
академічної групи _____126м-22-1_____
(шифр)
спеціальності _____126 « Інформаційні системи та технології»_____
(код і назва спеціальності)
за освітньо-професійною програмою _____

_____«Інформаційні системи та технології»_____
(офіційна назва)
на тему _____Розробка мобільного додатку для розміщення об'яв для продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter _____
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Олєвський В.І.			
розділів:				

Рецензент	д-р техн. наук І. С. Лактіонов			

Нормоконтролер	проф. Коротенко Г.М..			

Дніпро
2023

ЗАТВЕРДЖЕНО:
завідувач кафедри

___інформаційних технологій___
___та комп'ютерної інженерії____

(повна назва)

_____ __Гнатушенко В.В.__
(підпис) (прізвище, ініціали)

«_____»_____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня ____магістр _____

(бакалавра, спеціаліста, магістра)

студенту _____Волошину Д.А_____ **академічної групи** ____126м-22-1___
(прізвище та ініціали) (шифр)

спеціальності ___126 « Інформаційні системи та технології »_____

за освітньою-професійною програмою _____

_____«Інформаційні системи та технології»_____

на тему ____Розробка мобільного додатку для розміщення об'яв для продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від__09.10.23__№ 1227-С

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз теми та постановка задачі	1.10.2023 – 31.10.2023
Розділ 2	Створення архітектури та основних функціональних елементів додатку для розміщення об'яв з продажу транспортних засобів із гнучким алгоритмом фільтрації	1.11.2023 – 30.11.2023
Розділ 3	Розробка інформаційної системи мобільного кросплатформного додатку із гнучким алгоритмом фільтрації для створення об'яв з продажу транспортних засобів та запчастин	1.12.2023 – 10.12.2023

Завдання видано _____ _____Коротенко Г.М._____
(підпис керівника) (прізвище, ініціали)

Дата видачі _____1.10.2023 р._____

Дата подання до екзаменаційної комісії ____18.12.2023 р.____

Прийнято до виконання _____ _____Волошин Д.А_____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 79 стор., 53 рис., 3 додатки, 18 джерел.

Об'єкт дослідження: процеси створення кросплатформних додатків та світові тенденції у ринках ТЗ та мобільних додатків.

Предмет дослідження: інформаційні технології побудови кросплатформного додатку для створення об’яв з продажу ТЗ та запчастин до них.

Мета магістерської роботи: створення інформаційної системи кросплатформного додатку для створення об’яв з продажу ТЗ та запчастин до них на базі фреймворку Flutter.

У вступі подано стан проблеми та виконана постановка задачі дослідження.

Перший розділ присвячено аналізу тему дослідження та постановці задачі. Наведено огляд ринків ТЗ в Україні та світі і сучасні світові тенденції у розробці мобільних додатків.

В другому розділі наведено проектну складову вирішення завдання. Розглянуто деякі відомі методи побудови архітектури інформаційної системи кросплатформного додатку а також струтура бази даних для нього. Обґрунтовано вибір інструментів програмної реалізації інформаційної системи.

Третій розділ присвячено розробці інформаційної системи кросплатформного додатку для створення об’яв з продажу ТЗ та запчастин до них на мові Flutter. Виконано реалізацію інформаційної технології.

Практична цінність результатів полягає у тому, що розроблена інформаційна система кросплатформного додатку надає можливість використання платформи для створення об’яв з продажу ТЗ та запчастин до них.

МОВА FLUTTER, РИНОК ТРАНСПОРТНИХ ЗАСОБІВ, ФІЛЬТРАЦІЯ, КРОСПЛАТФОРМНІСТЬ

ABSTRACT

Explanatory note: 79 pages, 53 figures, 3 appendices, 18 sources.

Object of research: processes of creating cross-platform applications and global trends in the markets of vehicles and mobile applications.

Subject of research: information technologies for building a cross-platform application for creating advertisements for the sale of vehicles and spare parts for them.

The purpose of the master's thesis: to create an information system for a cross-platform application for creating advertisements for the sale of vehicles and spare parts based on the Flutter framework.

The introduction presents the state of the problem and states the research problem.

The first section is devoted to the analysis of the research topic and formulation of the problem. An overview of the vehicle markets in Ukraine and the world and current global trends in the development of mobile applications are presented.

The second chapter presents the design component of solving the problem. Some well-known methods for constructing the information system architecture of a cross-platform application, as well as the database structure for it, are considered. The choice of tools for software implementation of the information system is justified.

The third section is devoted to the development of an information system for a cross-platform application for creating advertisements for the sale of vehicles and spare parts in Flutter. The implementation of information technology has been completed.

The practical value of the results lies in the fact that the developed cross-platform application information system makes it possible to use the platform to create advertisements for the sale of vehicles and spare parts for them.

FLUTTER LANGUAGE, VEHICLE MARKET, FILTRATION, CROSS-PLATFORM

ЗМІСТ

Перелік умовних позначень

БД – база даних

ТЗ – транспортний засіб

Тоу – об’ява з транспортним засобом

Mod – об’ява з запчастиною

ПК – персональний комп’ютер

IOS – мобільна платформа

API – application programming interface

HTTP – Hyper Text Transfer Protocol

RAM – random access memory

SQL – Structured Query Language

JSON – JavaScript Object Notation

Вступ

Актуальність роботи. В контексті швидко ростучого попиту на транспортні засоби в Україні та світі, а відповідно і розвитку ринку та глобалізації, розглядаються питання для створення майданчиків спрямованих на локальні групи людей, котрі зацікавлені у збуті або придбанні запчастин та транспортних засобів, або у їх продажі. Люди постійно шукають доступні запчастини для своїх ТЗ, щоб здійснювати ремонт чи покращувати їхній вигляд і функціональність.

На сьогодення найбільшою популярністю для збуту або покупки будь-яких ТЗ та запчастин до них користуються є інтернет-портали та платформи, такі як OLX, Prom.ua, або інші інтернет портали в залежності від регіону. Такий хід речей обумовлений факторами:

Широка аудиторія:

Великі портали мають величезну користувацьку базу, що може забезпечити велику кількість переглядів оголошень.

Користувачі вже звикли використовувати ці портали для різних потреб.

Доступність через веб-браузер:

Інтернет-портали можна відвідувати з будь-якого пристрою з доступом до Інтернету через веб-браузер.

Відгуки та рейтинги:

Багато порталів надають можливість користувачам залишати відгуки та ставити рейтинг продавцям, що дозволяє покупцям зробити більш обдуманий вибір.

Однак існує мала кількість мобільних додатків які б виконували схожі функції, але й використовували переваги платформи які дають їм значну перевагу перед інтернет-ресурсами. В першу чергу найпомітнішими перевагами є: зручність та легкість використання, швидкий доступ, використання функцій пристрою, оптимізація для платформ, глибша інтеграція з операційним середовищем.

Тому буде актуально розробити кросплатформний додаток, котрий буде використовувати переваги пов’язані із платформами мобільних пристроїв та дослідити його переваги перед інтернет-ресурсами, а для реалізації кросплатформності буде використано такий інструментарій як Flutter Flow та мову програмування Flutter .

Об'єкт дослідження: процеси створення кросплатформних додатків та світові тенденції у ринках ТЗ та мобільних додатків.

Предмет дослідження: інформаційні технології побудови кросплатформного додатку для створення об’яв з продажу ТЗ та запчастин до них.

Мета роботи. Мета роботи полягає у створенні інформаційної системи кросплатформного додатку для для створення об’яв з продажу ТЗ та запчастин до них на базі фреймворку Flutter. Відповідно до мети й предмета дослідження у кваліфікаційній роботі необхідно вирішити наступні завдання:

- дослідити ринок мобільних додатків у світі та порівняти переваги та недоліки веб-сайтів із перевагами та недоліками мобільних додатків;

- дослідити ринок ТЗ в світі та Україні;

- розробити структуру нереляційної бази даних Firebase;

- розробити простий і зрозумілий інтерфейс для інформаційної системи додатку;

- розробити кросплатформний мобільний додаток із гнучким алгоритмом фільтрування, для створення об’яв з продажу ТЗ та запчастин до них.

Практична цінність результатів полягає у тому, що розроблена інформаційна система кросплатформного додатку надає можливість використання платформи для створення об’яв з продажу ТЗ та запчастин до них.

РОЗДІЛ 1
АНАЛІЗ ТЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

1. Основні поняття і технологій, що використовувалися для реалізації кросплатформного додатку.

Мобільний додаток згідно з [1] (також відомий як мобільний застосунок або просто "app") - це програмне забезпечення, спеціально розроблене для використання на мобільних пристроях, таких як смартфони чи планшети. Мобільні додатки дозволяють користувачам використовувати різноманітні функції та сервіси на своїх пристроях, зазвичай шляхом завантаження та встановлення їх з магазинів додатків, таких як App Store для iOS або Google Play для Android.

1.1. Історія розвитку мобільних додатків.

Історія розвитку мобільних додатків є динамічною. Цей етап в історії інформаційних технологій взяв свій початок разом з появою мобільних телефонів та їхнім подальшим розвитком. Спочатку мобільні телефони використовувалися переважно для дзвінків та повідомлень, і лише з часом з'явилися перші додатки. У 90-х роках минулого століття виникли прості ігри та утиліти для мобільних телефонів з використанням технології Java. Проте справжній прорив відбувся з появою смартфонів. Випуск iPhone від Apple в 2007 році визначив новий етап в розвитку, завдяки своєму інноваційному дизайну та App Store. Ключові чинники, що призвели до зростання популярності мобільних додатків, включають створення App Store та Google Play, покращення технічних характеристик мобільних пристроїв, розширення мережевого покриття, зростання популярності соціальних мереж та мультимедійного контенту, а також широкий вибір категорій додатків для задоволення різних потреб користувачів згідно з [2].

Сучасні мобільні додатки стали невід'ємною частиною повсякденного життя, пропонуючи широкий спектр функцій і сервісів для розваг, освіти, роботи та спілкування. Розширення можливостей апаратного забезпечення та підвищення швидкості мобільних мереж продовжують забезпечувати потужний та зручний інструментарій для користувачів та розробників.

1.1.2. Фактори, що спричинили ріст популярності мобільних додатків

App Store та Google Play**:** Запуск App Store і Google Play від Apple і Google відповідно в 2008 році дав можливість розробникам легко розповсюджувати та продавати свої додатки. Це зробило процес отримання та встановлення додатків простішим і зручнішим для користувачів.

Зростання потужності апарату: З плином часу мобільні пристрої стали потужнішими, з більш високоякісними процесорами, більшою кількістю оперативної пам'яті та великими екранами. Це дозволило створювати більш потужні та функціональні додатки.

Розвиток мереж: З впровадженням швидших мобільних мереж, таких як 3G та 4G, користувачі отримали швидкий та надійний доступ до Інтернету прямо зі своїх пристроїв. Це відкрило нові можливості для розвитку онлайн-сервісів та додатків.

Соціальні мережі та медіа: Зростання популярності соціальних мереж та мультимедійних контентів (фото, відео) виникло разом з розвитком мобільних додатків. Люди швидко перейшли від використання комп'ютерів до мобільних пристроїв для спільного використання контенту та отримання новин.

Широкий вибір категорій: Різноманітність категорій мобільних додатків дозволила задовольняти потреби різних груп користувачів. Від ігор і соціальних мереж до освітніх та бізнес-додатків, кожен може знайти щось для себе.

Розробка для більше ніж однієї платформи: З'явилися технології, що дозволяють розробляти додатки для кількох платформ, таких як iOS та

Android, за допомогою загального коду. Це спростило роботу розробників та збільшило охоплення аудиторії.

Інтеграція з апаратним забезпеченням: Можливість взаємодії з різними аспектами апаратного забезпечення, такими як камера, геолокація, акселерометр, датчики, відкрила нові можливості для розробників, щоб створювати інноваційні додатки.

1.2. Розвиток та становлення ринку транспортних засобів у світі та Україні після появи інтернету

Розвиток ринку транспортних засобів став тісно пов'язаний з появою та розвитком Інтернету. Цей процес переживав історичні та технологічні трансформації, які суттєво вплинули на способи покупки, продажу та використання транспортних засобів.

Етапи розвитку ринку транспортних засобів в контексті Інтернету:

Поява перших веб-сайтів:

З початком 1990-х років, коли Інтернет став доступним для широкого загалу, з'явилися перші веб-сайти, що надавали інформацію про продаж автомобілів. Однак це були переважно статичні ресурси, і велика частина торгівлі відбувалася традиційними методами.

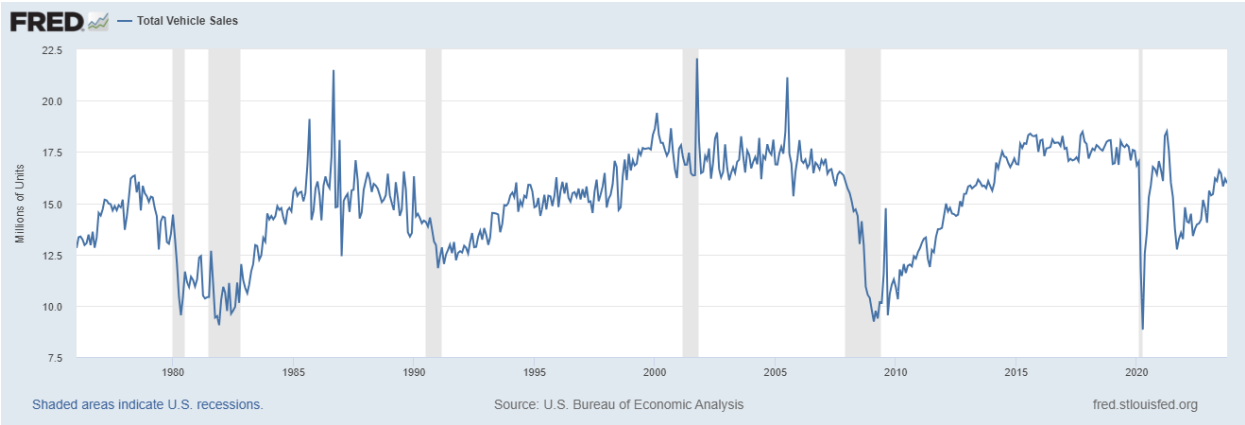

Рис.1.1. Загальний графік світових продаж ТЗ

На (Рис.1.1.) зображений графік [3] що демонструє кількість продаж ТЗ у світі, на ньому ми можемо побачити, що після спаду 1987-1993 років, почався активний ріст продажів пов’язаний із появою інтернету.

Онлайн-портали і класифіковані оголошення:

У другій половині 1990-х років з'явилися перші онлайн-портали та ресурси, спеціалізовані на класифікованих оголошеннях про продаж транспортних засобів. Це дало користувачам можливість розміщувати оголошення та переглядати пропозиції в Інтернеті.

Електронні аукціони:

З появою технологій онлайн-торгівлі, таких як електронні аукціони, процес купівлі-продажу автомобілів став більш ефективним та доступним. Різні платформи дозволяли учасникам придбати та продати автомобілі за допомогою Інтернет-технологій.

Розвиток онлайн-автосалонів:

У 2000-х роках почали з'являтися онлайн-автосалони, які надають повноцінні сервіси покупки автомобілів в Інтернеті. Клієнти мали змогу вибрати та придбати авто, обираючи з великого асортименту, і здійснювати фінансові операції онлайн.

Мобільні додатки та Інтернет в кишені:

Зі зростанням популярності смартфонів та планшетів з'явилися мобільні додатки для пошуку та купівлі автомобілів. Користувачі можуть швидко переглядати пропозиції, отримувати сповіщення про нові оголошення та навіть здійснювати оплату через мобільні додатки.

Аналітика та огляди:

Інтернет надав можливість користувачам отримати докладні відгуки, огляди та аналітику щодо конкретних моделей та брендів автомобілів. Це сприяло більш обдуманим рішенням покупців.

Електронна реєстрація та фінансування:

Нові технології дозволяють клієнтам проводити електронну реєстрацію та отримувати фінансування онлайн, спрощуючи весь процес покупки автомобіля.

Інтернет суттєво змінив парадигму транспортного ринку, забезпечуючи більше можливостей для покупців і продавців, розширюючи доступ до

інформації та полегшуючи сам процес торгівлі автотранспортом. Зараз Інтернет виступає ключовим фактором в утворенні та розвитку ринку транспортних засобів згідно з [4].

1.2.1. Розвиток, стан та перспективи ринку транспортних засобів в Україні

Розвиток (1990-2000 роки)

Починаючи з отримання незалежності у 1991 році, Україна вступила в еру глибоких трансформацій, охоплюючи всі галузі, зокрема й ринок транспортних засобів.

Початок Незалежності

Етап після отримання незалежності характеризувався змінами у всіх аспектах суспільства, включаючи ринок транспортних засобів. З лібералізацією економіки в 1990-ті роки ринок транспортних засобів став більш відкритим для імпорту, що викликало інтенсивну конкуренцію. Запровадження ринкових реформ призвело до збільшення доступності імпортних автомобілів, що розширило вибір для споживачів. У 2000-х роках спостерігалося активне зростання попиту на якісні та доступні автомобілі, внаслідок чого відбувалася ринкова адаптація.

Сучасний стан (починаючи з 2010-х років)

Останнє десятиріччя в Україні призначене новим тенденціям та інноваціям на ринку транспортних засобів. У 2010-х роках відбулося зростання внутрішнього виробництва автомобілів. Українські компанії активніше взяли участь на внутрішньому та зовнішньому ринках. Підвищення економічної активності та покращення фінансової здатності громадян стимулює модернізацію транспортного парку, зробивши споживачів більш вибагливими. За останні кілька років в Україні спостерігається зростання інтересу до екологічно чистих технологій, зокрема електричних та гібридних автомобілів.

Значущий вплив на ринок має розширення орендних та каршерінгових сервісів, що змінюють уявлення про користування автомобілями.

Перспективи

Зазирнемо в майбутнє, визначаючи ключові напрямки, які можуть визначити подальший розвиток ринку транспортних засобів в Україні. Подальший розвиток дорожньої та зарядної інфраструктури сприятиме збільшенню популярності електричних та гібридних автомобілів. Підтримка вітчизняних виробників та стимулювання внутрішнього виробництва може підняти якість та конкурентоспроможність українських автомобілів.

Впровадження новітніх технологій у транспортні засоби, таких як системи безпеки та допомоги водієві, буде важливим аспектом подальшого розвитку.

В цілому, ринок транспортних засобів в Україні знаходиться на шляху сталого розвитку, а його майбутнє формується під впливом багатьох факторів, від економічних до технологічних згідно з [5].

Із розвитком ринку транспортних засобів, розвитком технологій та глобалізацією утворилась необхідність створення засобів для розміщення оголошень про продаж транспортних засобів та запчастин є результатом швидкого розвитку інтернет-технологій та змін у споживчих звичках. Ці інновації відкрили нові можливості для власників ТЗ, покупців і продавців, створюючи зручний та ефективний спосіб обміну інформацією та укладання угод.

На сьогоднішній день в Україні існує кілька популярних інтернет-платформ та для продажу транспортних засобів та запчастин згідно з [6]:

OLX є однією з найбільших інтернет-платформ в Україні для розміщення оголошень про продаж різних товарів, включаючи транспортні засоби та запчастини. Сервіс дозволяє користувачам завантажувати фотографії, додавати детальний опис та взаємодіяти через систему відгуків.

Prom.ua володіє великою аудиторією та розглядається як торгова платформа для різних товарів та послуг. Вона також дозволяє підприємствам та приватним особам розміщувати свої транспортні засоби та запчастини.

Auto.ria.com - спеціалізований ресурс, фокусований саме на автомобільному секторі. Сервіс надає можливість розміщення оголошень про продаж автомобілів, мотоциклів, комерційного транспорту та запчастин. Він також надає інформацію про автомобільний ринок, новини та поради.

Aviso.ua - це ще один популярний ресурс для розміщення оголошень про продаж різних товарів, включаючи транспортні засоби. Платформа пропонує великий вибір автомобілів та запчастин від різних власників.

1.3. Переваги та недоліки мобільних додатків перед інтернет ресурсами

Мобільні додатки мають ряд переваг перед сайтами, які роблять їх більш ефективним інструментом для бізнесу.

Мобільні додатки, як правило, мають більш зручний та інтуїтивно зрозумілий інтерфейс, ніж веб-сайти. Це пов'язано з тим, що вони розроблені спеціально для використання на мобільних пристроях. Мобільні додатки можуть використовувати функції, такі як жестове керування, віджети та push-повідомлення, які неможливо або складно реалізувати на веб-сайтах.

Мобільні додатки можуть використовувати унікальні функції, які неможливо реалізувати на веб-сайтах. До таких функцій належать:

Зручність та інтуїтивність

Мобільні додатки мають ряд переваг перед веб-сайтами, зокрема більш зручний та інтуїтивно зрозумілий інтерфейс. Це пов'язано з тим, що мобільні додатки розроблені спеціально для використання на мобільних пристроях, які мають обмежений розмір екрану та інші особливості. Веб-сайти, з іншого боку, повинні бути адаптивними, щоб відображатися коректно на різних пристроях, включаючи ПК, ноутбуки, планшети та смартфони. Це може

призвести до того, що веб-сайти виглядатимуть менш зручними та інтуїтивно зрозумілими на мобільних пристроях.

Жестове керування є однією з ключових переваг мобільних додатків. Жестове керування - це спосіб взаємодії з мобільним пристроєм за допомогою рухів пальців. Наприклад, користувач може провести пальцем по екрану, щоб прокрутити вгору або вниз, або двома пальцями, щоб збільшити або зменшити масштаб. Жестове керування робить взаємодію з мобільним додатком більш природною та інтуїтивно зрозумілою.

Віджети - це ще одна перевага мобільних додатків. Віджети - це маленькі програми, які можна додати на головний екран мобільного пристрою. Віджети можуть надавати користувачам швидкий доступ до інформації або функцій, таких як прогноз погоди, новини або календар. Віджети роблять мобільні додатки більш доступними та корисними.

Push-повідомлення - це ще одна перевага мобільних додатків. Push-повідомлення - це спосіб для бізнесу спілкуватися з користувачами безпосередньо на їхніх мобільних пристроях. Push-повідомлення можна використовувати для інформування користувачів про нові продукти, послуги та події. Push-повідомлення можуть допомогти бізнесу залучити нових клієнтів та підвищити продажі.

Персоналізований контент і пропозиції.

Окрім цих функцій, мобільні додатки також можуть використовуватися для персоналізації контенту та пропозицій для користувачів. Це можна зробити, використовуючи дані, які користувачі надають при встановленні та використанні додатка. Наприклад, додаток для покупок може використовувати історію покупок користувача, щоб показати йому пропозиції на товари, які він, ймовірно, зацікавиться.

Персоналізований контент і пропозиції можуть допомогти бізнесу поліпшити взаємодію з клієнтами та підвищити продажі.

Загалом, мобільні додатки мають ряд переваг перед веб-сайтами, які роблять їх більш ефективним інструментом для бізнесу. Ці переваги

включають більш зручний та інтуїтивно зрозумілий інтерфейс, використання жестового керування та віджетів, а також можливість персоналізації контенту та пропозицій. Геолокація: мобільні додатки можуть використовувати геолокацію для надання користувачам персоналізованого контенту та пропозицій.

Мобільні платежі: мобільні додатки можуть використовувати мобільні платежі для спрощення покупки товарів та послуг.

Мобільні додатки можуть використовувати дані, які користувачі надають при встановленні та використанні додатка, для персоналізації контенту та пропозицій. Це може допомогти бізнесу поліпшити взаємодію з клієнтами та підвищити продажі.

Мобільні додатки доступні користувачам в будь-який час і в будь-якому місці, де є мобільний зв'язок. Це може бути особливо важливим для бізнесу, який працює в сфері послуг або розваг.

Вартість розробки та підтримки мобільного додатка може бути вищою, ніж вартість розробки та підтримки веб-сайту. Однак, з урахуванням переваг, які надають мобільні додатки, ця різниця в ціні може бути виправданою для багатьох бізнесів.

Мобільна реклама є ефективним способом охоплення цільової аудиторії. Мобільні додатки можуть використовуватися для показу мобільної реклами, яка є більш релевантною та таргетованою, ніж реклама на веб-сайтах.

Мобільні додатки мають ряд переваг перед сайтами, які роблять їх більш ефективним інструментом для бізнесу. Якщо розглядається можливість розробки мобільного додатку для свого бізнесу, ці переваги повинні переконати вас у тому, що це правильне рішення.

У 2023 році 67,9% світового населення [7] буде використовувати мобільні пристрої. Це означає, що більшість людей у світі матимуть доступ до мобільних додатків.

У 2023 році 59% покупок [7] в Інтернеті буде здійснено через мобільні пристрої. Це означає, що мобільні додатки є основним каналом для здійснення покупок в Інтернеті.

У 2023 році 73% мобільних [7] користувачів будуть отримувати push-повідомлення від бізнесу.

Ці дані свідчать про те, що мобільні пристрої є важливим каналом для бізнесу. Бізнес, який хоче досягти успіху в Інтернеті, повинен мати присутність на мобільних пристроях.

Мобільні додатки мають ряд переваг перед сайтами, які роблять їх більш ефективним інструментом для бізнесу. Вони забезпечують більш зручний та інтуїтивно зрозумілий інтерфейс, унікальні функції, доступність у будь-який час і в будь-якому місці, а також можливість використання даних для персоналізації контенту та пропозицій. Крім того, мобільні додатки є ефективним способом охоплення цільової аудиторії та проведення маркетингових кампаній.

1.4 Переваги інтернет ресурсів

Веб-сайти мають ряд переваг перед мобільними додатками, які роблять їх більш ефективним інструментом для бізнесу в деяких випадках.

1) Доступність

Веб-сайти доступні на всіх пристроях, які мають доступ до Інтернету. Це означає, що користувачі можуть отримати доступ до веб-сайту з будь-якого комп'ютера, ноутбука, планшета або смартфона. Мобільні додатки, з іншого боку, доступні лише на пристроях, на яких вони були встановлені.

2) Розробка та підтримка

Веб-сайти, як правило, дешевші та простіші у розробці та підтримці, ніж мобільні додатки. Це пов'язано з тим, що веб-сайти не вимагають розробки для різних операційних систем.

3) Інтеграція з іншими системами

Веб-сайти можна легко інтегрувати з іншими системами, такими як CRM, ERP та маркетинг. Це може допомогти бізнесу поліпшити ефективність своїх операцій.

4) Веб-маркетинг

Веб-сайти можна просувати за допомогою різних веб-маркетингових каналів, таких як пошукова оптимізація, соціальні мережі та контекстна реклама. Це може допомогти бізнесу залучити нових клієнтів.

5) Доступ до інформації

Веб-сайти можуть містити більше інформації, ніж мобільні додатки. Це пов'язано з тим, що веб-сайти не обмежені розміром екрану мобільного пристрою.

6) Соціальні функції

Веб-сайти можуть включати соціальні функції, такі як коментарі, лайки та поширення. Це може допомогти бізнесу взаємодіяти з клієнтами та створювати спільноту.

7) Вимоги стандартам

Веб-сайти повинні відповідати стандартам веб-доступності. Це означає, що вони повинні бути доступні для людей з обмеженими можливостями. Мобільні додатки не обов'язково повинні відповідати стандартам веб-доступності.

У 2023 році 67,9% світового населення буде використовувати мобільні пристрої згідно [7]. Однак, не всі користувачі мобільних пристроїв встановлюють мобільні додатки. За даними Statista, у 2023 році 62,6% мобільних користувачів у США будуть використовувати мобільні додатки.

Вартість розробки та підтримки мобільного додатка може бути вищою, ніж вартість розробки та підтримки веб-сайту. Однак, вартість розробки та підтримки мобільного додатка може бути виправданою для бізнесу, який потребує спеціальних функцій або можливостей, які неможливо реалізувати на веб-сайті.

Веб-сайти можуть бути просунуті за допомогою різних веб-маркетингових каналів, таких як пошукова оптимізація, соціальні мережі та контекстна реклама. Це може допомогти бізнесу залучити нових клієнтів та підвищити продажі.

Висновок:

Веб-сайти та мобільні додатки мають свої переваги та недоліки. Вибір між веб-сайтом та мобільним додатком залежить від конкретних потреб бізнесу.

Веб-сайти мають ряд переваг перед мобільними додатками, які роблять їх більш ефективним інструментом для бізнесу в деяких випадках. Ці переваги включають доступність, розробку та підтримку, інтеграцію з іншими системами, веб-маркетинг, доступ до інформації, соціальні функції, відповідність стандартам та мобільні версії.

Однак, мобільні додатки також мають ряд переваг, таких як зручний та інтуїтивно зрозумілий інтерфейс, використання жестового керування та віджетів, а також можливість персоналізації контенту та пропозицій.

Вибір між веб-сайтом та мобільним додатком залежить від конкретних потреб бізнесу. Якщо бізнес продовжує розвиватися та розширюватися, йому може знадобитися обидва інструменти.

Веб-сайт - це основа присутності бізнесу в Інтернеті. Він забезпечує основний контент і функціональність, які клієнти очікують від компанії. Мобільний додаток може надавати додаткові функції та можливості, які можуть бути корисними для клієнтів, такі як:

1) Зручний доступ до інформації та послуг в будь-який час і в будь-якому місці.

2) Персоналізований контент і пропозиції.

3) Мобільні платежі.

Ось деякі приклади того, як бізнес може використовувати як веб-сайт, так і мобільний додаток:

Роздрібний магазин: веб-сайт може надавати інформацію про продукти та послуги, а мобільний додаток може використовуватися для онлайн-покупок, відстеження замовлень та отримання спеціальних пропозицій.

Сервіс обслуговування клієнтів: веб-сайт може надавати інформацію про продукти та послуги, а мобільний додаток може використовуватися для подання скарг, отримання технічної підтримки та зв'язку з представниками служби підтримки клієнтів.

Соціальна мережа: веб-сайт може надавати основний контент і функціональність, а мобільний додаток може використовуватися для обміну повідомленнями, публікації контенту та доступу до соціальних функцій.

1.5. Капіталізація та популярність мобільних додатків

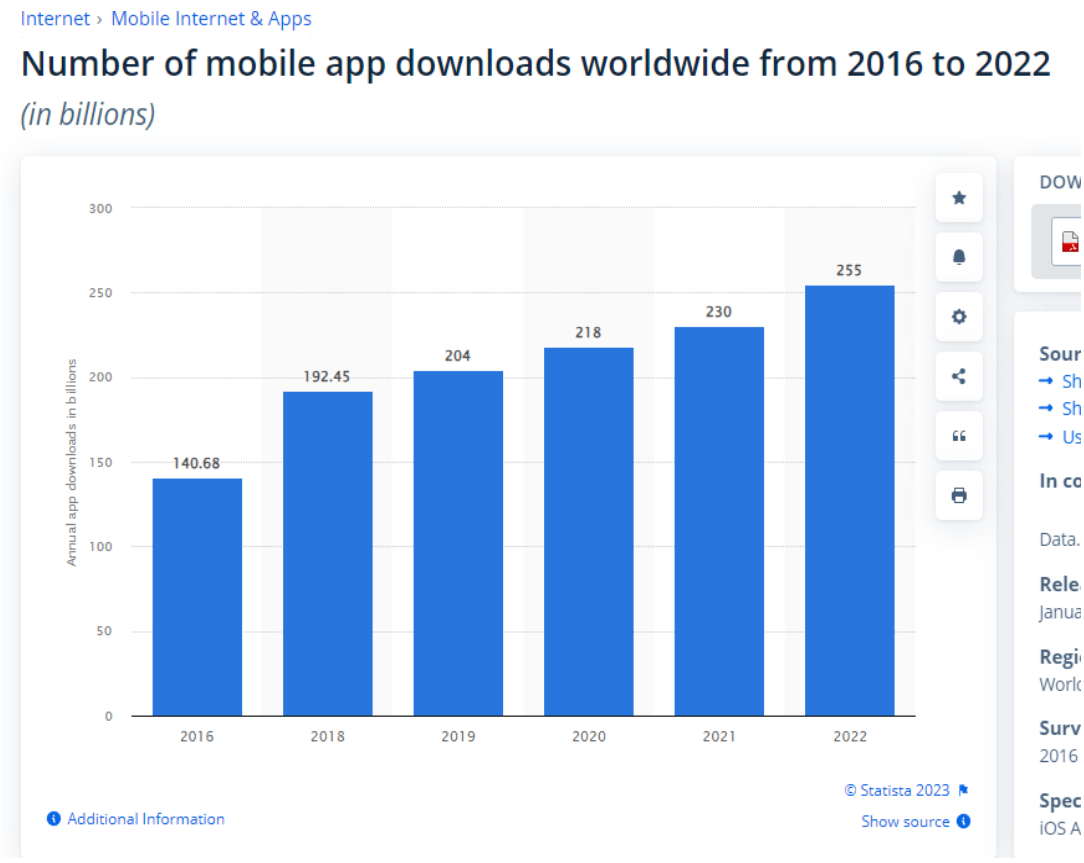

Рис.1.2. Зріст кількості завантажень [8] мобільних додатків 2016-2022 рр.

Прогнозується(рис.1.3) значне зростання показника «Завантаження» в усіх сегментах у 2027 році. Тенденція, що спостерігається з 2019 по 2027 роки, залишається незмінною протягом усього прогнозованого періоду.

Спостерігається постійне зростання показника по всіх сегментах. Примітно, що у 2027 році сегмент «Ігри» досяг найвищого значення — 176,1 мільярда завантажень [9].

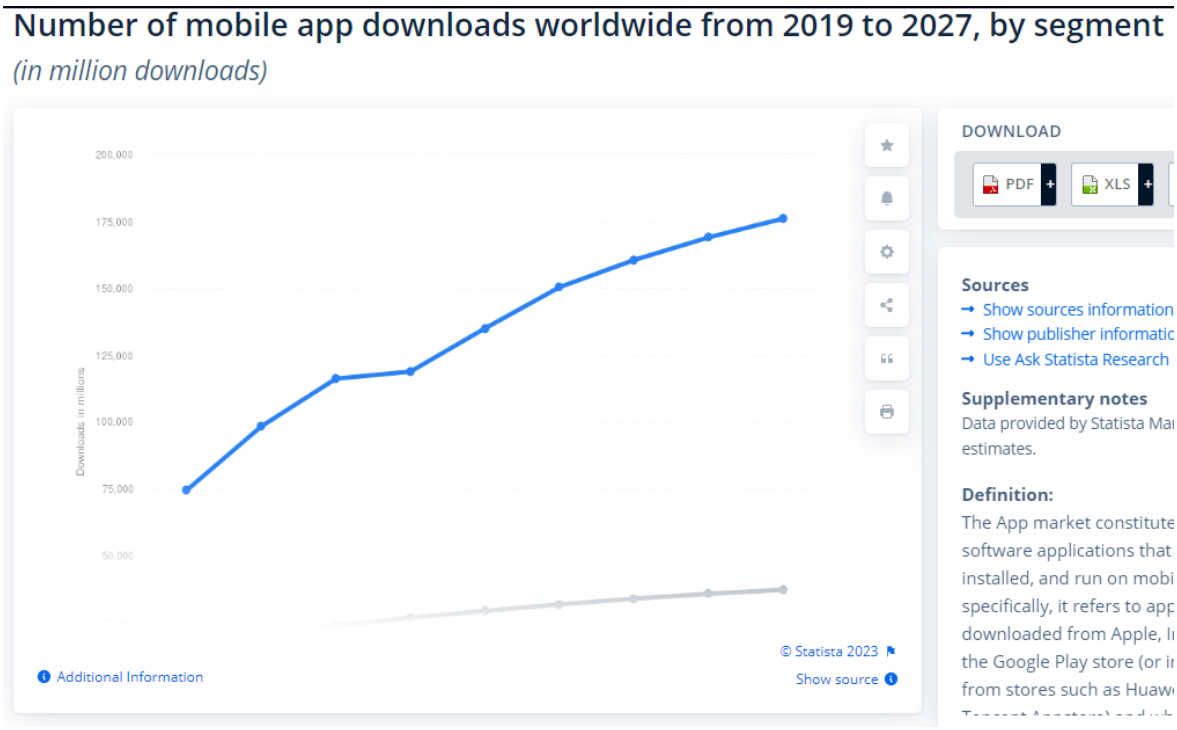

Рис.1.3. Прогноз на кількість завантажень з 2019 по 2027

За прогнозами, у 2025 році мобільні додатки принесуть понад 613 мільярдів доларів доходу, причому мобільні ігри становлять найбільшу частку доходу серед усіх категорій додатків. У 2020 році ігри та відео становили найбільшу частку ринку мобільного контенту за рік. Сектори електронного видавництва та освіти все ще мали обмежений ринок для свого мобільного контенту, незважаючи на збільшення використання додатків, викликане пандемією COVID-19, яка порушила звичайні налаштування шкільної системи.

Як невід'ємну частину роботи зі смартфоном, найбільшу кількість додатків у основних магазинах додатків можна завантажити безкоштовно. Однак за останні роки зростання споживчих витрат у всьому світі на програми продемонструвало здоровий апетит користувачів до преміум-сервісів або контенту платних програм. У другому кварталі 2021 року споживачі Android

витратили в середньому 5,31 долара США на телефон після піку в останньому кварталі 2020 року, коли в середньому 10,6 долара США на мобільний пристрій. Станом на вересень 2021 року кількість платних додатків скоротилася до шести та чотирьох відсотків від загальної кількості в Apple App Store та Google Play Store відповідно. Для порівняння, програми з планами передплати стають дедалі популярнішими в сфері монетизації. У 2020 році провідні програми за підпискою в Apple App Store заробили понад 10 мільйонів доларів США світового доходу.

1.6. Технології обрані для створення мобільного додатку

1.6.1. Фреймворк Flutter

Рис.1.4. Логотип Flutter

Flutter - це відкритий фреймворк для розробки мобільних додатків, який був розроблений компанією Google. Flutter(рис.1.4) використовує єдиний код для створення додатків для Android, iOS, десктопних комп'ютерів та веб-браузерів [10].

Flutter має ряд особливостей, які роблять його привабливим для розробників мобільних додатків:

Єдиний код для всіх платформ: Flutter використовує єдиний код для створення додатків для Android, iOS, десктопних комп'ютерів та веб-браузерів.

Це робить розробку мобільних додатків більш ефективною та менш витратною.

Flutter використовує Dart - новий мову програмування, розроблену компанією Google. Dart - це мова програмування, яка поєднує в собі кращі риси Java та JavaScript. Dart є статичною мовою програмування, що означає, що всі помилки повинні бути виявлені до виконання. Dart також є динамічною мовою програмування, що означає, що змінні можуть бути змінені в процесі виконання. Flutter використовує візуальний компілятор, який дозволяє створювати високоефективні додатки. Візуальний компілятор перетворює код Dart у машинний код для конкретної платформи. Це дозволяє Flutter створювати додатки, які працюють швидко та ефективно. Flutter підтримує всі сучасні мобільні можливості, такі як сенсорне управління, геолокація та мобільні платежі. Flutter також підтримує деякі менш поширені мобільні можливості, такі як 3D-графіка та віртуальна реальність.

Використання

Flutter використовується для розробки широкого спектру мобільних додатків, включаючи[10]:

1) Соціальні мережі: Facebook, Twitter, Instagram

2) Електронна комерція: Amazon, eBay, Alibaba

3) Новини і погода: BBC, CNN, The Weather Channel

4) Ігри: Angry Birds, Candy Crush, Pokémon Go

Flutter також використовується для розробки мобільних додатків для корпоративного використання. Наприклад, компанія Google використовує Flutter для розробки свого мобільного додатка для співробітників.

Flutter постійно розвиваєтьсяб, у найближчих планах розробників Flutter:

1) Підтримка нових платформ: Flutter планує підтримувати нові платформи, такі як Wear OS, Tizen.

2) Покращення продуктивності: Flutter планує покращити продуктивність додатків, створених за допомогою Flutter .

3) Розширення функціональності: Flutter планує розширити функціональність, доступну для розробників.

Flutter - це потужний і універсальний фреймворк для розробки мобільних додатків. Він стає все більш популярним серед розробників, оскільки пропонує ряд переваг, таких як єдиний код для всіх платформ, швидкість і ефективність, а також підтримка сучасних мобільних можливостей.

Згідно з опитуванням розробників 2022 року, Flutter є найпопулярнішим кросплатформним мобільним фреймворком, який використовують глобальні розробники. Згідно з опитуванням, 46 відсотків розробників програмного забезпечення використовували Flutter . Загалом приблизно третина мобільних розробників використовує кросплатформні технології або фреймворки; решта мобільних розробників використовують рідні інструменти.

1.6.2. Кросплатформність

Кросплатформні мобільні фреймворки [11] використовуються для створення програми, доступної через велику кількість різноманітних кінцевих пристроїв. Один код використовується на кількох платформах для легкої переносимості, і в кінцевому підсумку вартість розробки програмного забезпечення залишається низькою. Завдяки кросплатформним мобільним фреймворкам є можливість максимально охопити цільову аудиторію. Наприклад, одна програма може бути націлена як на платформи iOS, так і на Android, що максимізує охоплення програми.

У 2020 році у світі було 24,5 мільйона(рис.1.5) розробників програмного забезпечення [12]. Очікується, що в найближчі роки кількість розробників зросте, особливо через зростання додатків і веб-розробок у зростаючу цифрову епоху. З усіх видів розробників програмного забезпечення розробники мобільних пристроїв є одними з найнижчих за шкалою оплати праці розробників. Мобільні розробники в середньому отримують річну зарплату 41 600 доларів США, тоді як інженерні менеджери є найбільш

високооплачуваними розробниками програмного забезпечення з середньорічною зарплатою 96 000 доларів США.

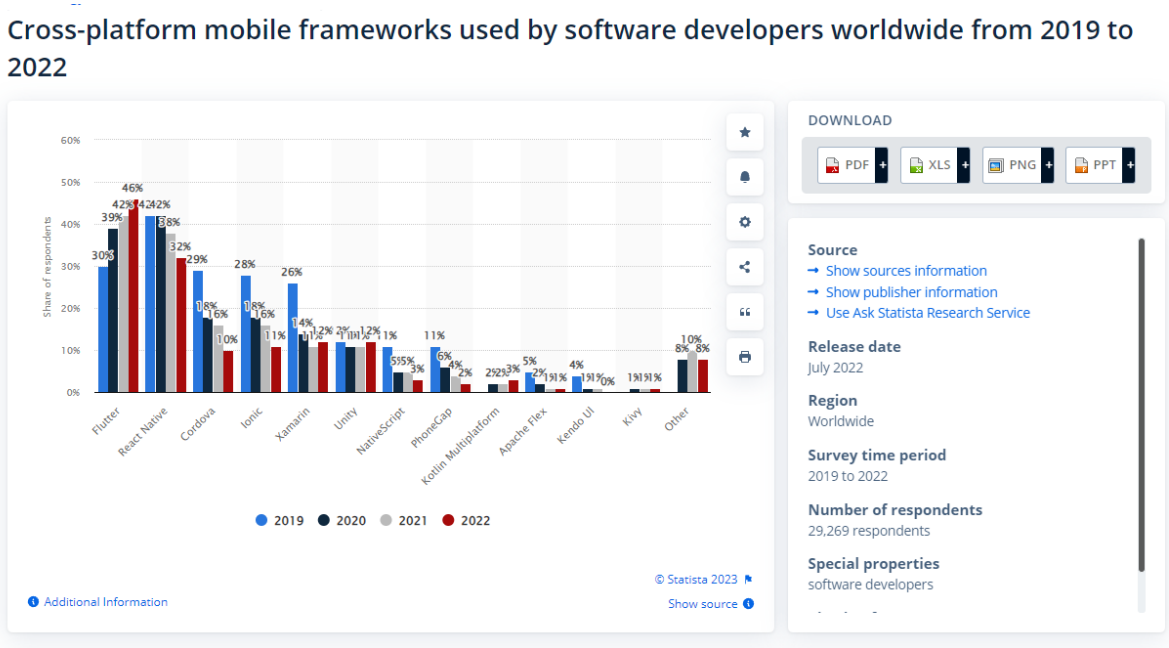

Рис.1.5. Популярність Flutter

Вибір фреймворку для розробки мобільних додатків

При виборі фреймворку для розробки мобільних додатків необхідно враховувати ряд факторів, таких як:

1) Можливості фреймворку: Фреймворк повинен підтримувати всі необхідні функції та можливості для розробки додатка.

2) Складність використання фреймворку: Фреймворк повинен бути достатньо простим у використанні, щоб ви могли швидко та ефективно розробити свій додаток.

3) Продуктивність фреймворку: Фреймворк повинен бути достатньо продуктивним, щоб ви могли створювати додатки, які працюють швидко та ефективно.

4) Доступність фреймворку: Фреймворк повинен бути доступним для розробників.

Flutter відповідає всім цим вимогам. Він підтримує широкий спектр функцій і можливостей, є відносно простим у використанні, продуктивний і доступний.

Переваги Flutter перед іншими фреймворками:

1) Flutter має ряд переваг перед іншими фреймворками для розробки мобільних додатків, таких як:

2) Єдиний код для всіх платформ: Flutter дозволяє розробляти додатки для всіх основних мобільних платформ за допомогою єдиного коду. Це економить час і гроші розробникам.

3) Швидкість і ефективність: Flutter використовує візуальний компілятор, який дозволяє створювати додатки, які працюють швидко та ефективно.

4) Підтримка сучасних мобільних можливостей: Flutter підтримує всі сучасні мобільні можливості, такі як сенсорне управління, геолокація та мобільні платежі.

Flutter також має деякі недоліки, такі як:

1) Нова мову програмування: Flutter використовує Dart, нову мову програмування, яка може бути складною для вивчення.

2) Не всі функції доступні для всіх платформ: Деякі функції Flutter доступні лише для певних платформ.

Висновок

Flutter - це потужний і універсальний фреймворк для розробки мобільних додатків. Він стає все більш популярним серед розробників, оскільки пропонує ряд переваг, таких як єдиний код для всіх платформ, швидкість і ефективність, а також підтримка сучасних мобільних можливостей.

Однак, Flutter не є ідеальним вибором для всіх проектів. Якщо ви не знайомі з Dart або вам потрібні певні функції, які доступні лише для певних платформ, вам слід розглянути інші фреймворки для розробки мобільних додатків.

1.6.3. FlutterFlow

Рис.1.6. Логотип FlutterFlow

FlutterFlow [13] - це потужна платформа(рис.1.6) без коду/low-коду, яка використовується для створення мобільних та веб-додатків. Вона дозволяє користувачам створювати інтерфейси користувачів з точним піксельним відтворенням та підключати їх до живих джерел даних за допомогою Firebase або API.

FlutterFlow стала популярною в останні роки, оскільки вона пропонує ряд переваг, які роблять її привабливим вибором для розробників мобільних та веб-додатків. Зокрема, вона:

Проста у використанні: FlutterFlow використовує інтерфейс перетягування та відпускання, який робить її простим у використанні для початківців. Це означає, що навіть люди без досвіду розробки програм можуть створювати високоякісні додатки за короткий час.

Широкий спектр вбудованих компонентів та функцій: FlutterFlow включає в себе широкий спектр вбудованих компонентів та функцій, які можуть допомогти розробникам швидко створювати повнофункціональні додатки. Це може заощадити час та зусилля розробникам, які не хочуть писати код з нуля.

Безшовна інтеграція з Firebase: FlutterFlow інтегрується з Firebase, що забезпечує доступ до широкого спектру функціональних можливостей, таких як аутентифікація користувачів, зберігання даних і функції в реальному часі. Це може допомогти розробникам створювати більш складні додатки з меншими зусиллями.

Можливість розширювати функціональність за допомогою власного коду: FlutterFlow дозволяє розробникам розширювати функціональність своїх додатків за допомогою власного коду. Це надає розробникам гнучкість, необхідну для створення додатків з унікальними вимогами.

FlutterFlow також має деякі недоліки, які слід враховувати перед її вибором.

Обмежена гнучкість порівняно з розробкою рідних додатків: FlutterFlow не надає такої ж гнучкості, як розробка рідних додатків, оскільки вона обмежена функціональністю, яку надає Firebase.

Крива навчання для складних функціональностей: Створення складного функціоналу у FlutterFlow може бути обмежена, оскільки це вимагає розуміння мови Flutter (Dart) та її екосистеми та сам функціонал не достаньо гнучкий, не впроваджує повного використання сторонніх бібліотек.

Незважаючи на ці недоліки, FlutterFlow є потужною платформою, яка може бути корисною для розробників мобільних та веб-додатків. Вона є хорошим вибором для початківців і досвідчених розробників, які хочуть створювати високоякісні додатки швидко та ефективно.

FlutterFlow підтримує розробку мобільних і веб-додатків для Android, iOS і вебу. Це означає, що розробники можуть використовувати одну платформу для створення додатків для різних платформ.

Експортовані програми можна розгорнути на різних платформах хостингу, таких як Firebase Hosting. Це дозволяє розробникам розгорнути свої додатки на будь-якій платформі, яка відповідає їхнім потребам.

FlutterFlow пропонує активну спільноту для підтримки та обміну знаннями. Це може бути корисним для розробників, які потребують допомоги або хочуть дізнатися більше про FlutterFlow.

FlutterFlow - чудовий вибір для створення високоякісних мобільних і веб-додатків швидко та ефективно. Він підходить для початківців і досвідчених розробників. Однак важливо враховувати складність проекту перш ніж вибирати FlutterFlow.

Одже, FlutterFlow має функціонал, здатний суттєво полегшити більшість виробничго процесу розробки мобільного додатку, такі як: створення UI-елементів додатку, написання структур даних, автоматизоване налаштування Firebase, Stripe, авторизації у додаток та інш. Незважаючи на переваги у цього інструменту також є певні мінуси, що можуть стати критичними при розробці додатку, основним із них є неможливість створення вкладеності бази даних більш ніж одна підколекція, створити більш гнучку фільтрацію даних із Firebase та найважливіше – кешування зображень на локальному пристрої. У випадка, коли функціонал FlutterFlow більше нічого не може зробити із необхідними задачами проекту можна скористатися функцію вивантаження коду для подальшої розробки у IDE.

1.6.4 Нереляційна база даних Firebase

Рис.1.7. Логотип Firebase

Firebase [14] – це продукт Google(рис.1.7), який допомагає розробникам легко створювати, керувати та розвивати свої програми. Це допомагає розробникам створювати свої програми швидше та безпечніше. На стороні Firebase не потрібно програмування, що дозволяє більш ефективно використовувати його функції. Він надає послуги для Android, iOS, Інтернету та Unity. Він забезпечує хмарне сховище. Він використовує NoSQL для бази даних для зберігання даних.

Коротка історія Firebase:

Спочатку Firebase була постачальником послуг онлайн-чату для різних веб-сайтів через API і працювала під назвою Envolve. Він став популярним, оскільки розробники використовували його для обміну даними додатків, наприклад станом гри, в реальному часі між своїми користувачами, а не в чатах. Це спричинило поділ архітектури Envolve та її системи чату. Архітектура Envolve була розвинена її засновниками Джеймсом Темпліном та Ендрю Лі до того, чим є сучасна Firebase у 2012 році.

База даних Firebase Realtime - це хмарна база даних NoSQL, яка керує даними з неймовірною швидкістю в мілісекунди. Простіше кажучи, його можна як великий файл JSON.

Cloud Firestore: Хмарний Firestore — це база даних документів NoSQL, яка надає такі послуги, як зберігання, синхронізація та запити через програму у глобальному масштабі. Він зберігає дані у формі об'єктів, також відомих як документи. Він має пару ключ-значення і може зберігати всі види даних, наприклад, рядки, двійкові дані і навіть дерева JSON.

Аутентифікація: служба аутентифікації Firebase надає прості у використанні бібліотеки інтерфейсу користувача і SDK для аутентифікації користувачів у додатку. Це скорочує трудовитрати та зусилля, необхідні для розробки та обслуговування служби автентифікації користувачів. Він навіть справляється з такими завданнями, як об'єднання облікових записів, що якщо робити це вручну, може виявитися стомлюючим.

Дистанційне налаштування: служба віддаленої установки допомагає негайно публікувати оновлення для користувача. Зміни можуть змінюватись від зміни компонентів інтерфейсу користувача до зміни поведінки додатків. Вони часто використовуються при публікації сезонних пропозицій та контенту у додатку, термін дії якого обмежений.

Хостинг: Firebase забезпечує швидкий та безпечний хостинг додатків. Його можна використовувати для розміщення статичних або динамічних веб-

сайтів та мікросервісів. Він має можливість розмістити програму за допомогою однієї команди.

Хмарні повідомлення Firebase (FCM): служба FCM забезпечує з'єднання між сервером і кінцевими користувачами програми, яку можна використовувати для отримання та надсилання повідомлень та повідомлень. Ці з'єднання надійні та заощаджують заряд батареї.

Нижче наведено всі функції, необхідні для перевірки та керування перед офіційним запуском програми. Включені послуги:

Crashlytics: використовується для отримання звітів про збої в реальному часі. Ці звіти можна використовувати для покращення якості програми. Найцікавіша частина цього сервісу полягає в тому, що він дає докладний опис збою, який легше розробникам аналізувати.

Моніторинг продуктивності: ця послуга дає уявлення про характеристики продуктивності програм. SDK для моніторингу продуктивності можна використовувати для отримання даних про продуктивність із програми, їх перегляду та внесення відповідних змін до програми через консоль Firebase.

Тестова лабораторія: ця послуга допомагає тестувати програми на реальних та віртуальних пристроях, наданих Google та розміщених у центрах обробки даних Google. Це хмарна інфраструктура тестування програм, яка підтримує тестування програми на різних пристроях і конфігураціях пристроїв.

Розповсюдження додатків. Ця послуга використовується для попереднього випуску програм, які можуть бути протестовані довіреними тестувальниками. Це зручно, оскільки скорочується час, необхідний отримання зворотного зв'язку від тестувальників.

Аналітика Google: - безкоштовна служба вимірювання програм, що надається Google, яка дає представлений ні про використання додатків та взаємодію з користувачами. Він надає необмежені звіти з 500 різних автоматичних або користувальницьких подій з використанням Firebase SDK.

Firebase Predictions використовує машинне навчання для обробки аналітичних даних програми, додатково створюючи динамічні сегменти користувачів, що базуються на поведінці користувача. Вони автоматично доступні для використання у програмі через Firebase Remote Config, композитор сповіщень, обмін повідомленнями всередині програм Firebase та A/B-тестування.

Динамічні посилання: Deeps Links - це посилання, які направляють у перенаправляє користувача на певний контент. Firebase надає службу динамічних посилань, яка перетворює глибокі посилання на динамічні посилання, які можуть безпосередньо направляти користувача до зазначеного контенту всередині програми. Динамічні посилання використовуються для перетворення веб-користувачів на користувачів власних програм. Це також збільшує конверсію обміну даними між користувачами. Крім того, його також можна використовувати для інтеграції соціальних мереж, електронної пошти та SMS для підвищення залучення користувачів усередині програми[14].

A/B-тестування: воно використовується для оптимізації роботи програми за рахунок забезпечення його безперебійної роботи, масштабування продукту та проведення маркетингових експериментів.

Плюси та мінуси використання Firebase

Переваги:

1) Безкоштовні плани для новачків.

2) Доступна база даних у режимі реального часу.

3) Зростання спільноти.

4) Доступні численні послуги.

Недоліки:

Firebase використовує NoSQL, тому люди, що переходять із SQL, можуть зазнавати труднощів.

Зазвичай базу даних використовувати реального часу використовують як основне сховище, що не завжди добре. Основна проблема – обмежені можливості запитів. Ви не можете запитувати більше одного ключа за раз, і служба не надає спосіб фільтрації даних. Це тому, що вся БД є файлом JSON і

не має нічого спільного з форматом зберігання SQL. Формат також унеможливлює моделювання даних.

Обмежена міграція даних

Ви не розміщуєте дані, всі дані розміщуються на Firebase, і це серйозна проблема використання платформ BaaS як серверна частина програми. Якщо Firebase не надає інструмент міграції, що дозволяє легко переносити дані користувача, він сильно обмежує міграцію даних. Це робить користувачів залежними від платформи, і в разі зміни концепції серверної бази немає можливості перенести програму в інше джерело.

Підсумовуючи плюси та мінуси Firebase, можна сказати, що це гарний вибір, для створення нового продукту із нуля або що б переписати існуючий. Платформа допомагає зберігати та покращувати динамічний контент. Крім того, прискорює процес розробки, дозволяючи отримувати швидші результати — тож це ідеальний варіант для створення MVP додатку, котрий як раз і буде створений у ході виконаяння поставленої задачі.

Постановка завдання

- Створити мобільний кросплатформний додаток для розміщення об’яв з продажу ТЗ та деталей для них
- Розробити модель бази даних для додатку
- Провести тест навантаження
- Розробити алгоритм фільтрації що не навантажує пристрій та мережу
- Розробити зручний інтерфейс додатку

РОЗДІЛ 2
СТВОРЕННЯ АРХІТЕКТУРИ ТА ОСНОВНИХ ФУНКЦІОНАЛЬНИХ ЕЛЕМЕНТІВ ДОДАТКУ ДЛЯ РОЗМІЩЕННЯ ОБ'ЯВ З ПРОДАЖУ ТРАНСПОРТНИХ ЗАСОБІВ ІЗ ГНУЧКИМ АЛГОРИТМОМ ФІЛЬТРАЦІЇ

2.1. Архітектура додатку

Розробники [15] завжди віддають перевагу розробці програм із застосуванням шаблону архітектури програмного забезпечення. Шаблон архітектури надає файлам проекту модульність та гарантує, що всі коди будуть охоплені модульним тестуванням. Це спрощує завдання розробників з обслуговування програмного забезпечення та розширення функцій програми у майбутньому. MVC (модель - вистава - контролер), MVP (модель - вистава - презентатор) і MVVM (модель - вистава - ViewModel) - це найбільш популярний і визнаний у галузі шаблон архітектури Android серед розробників.

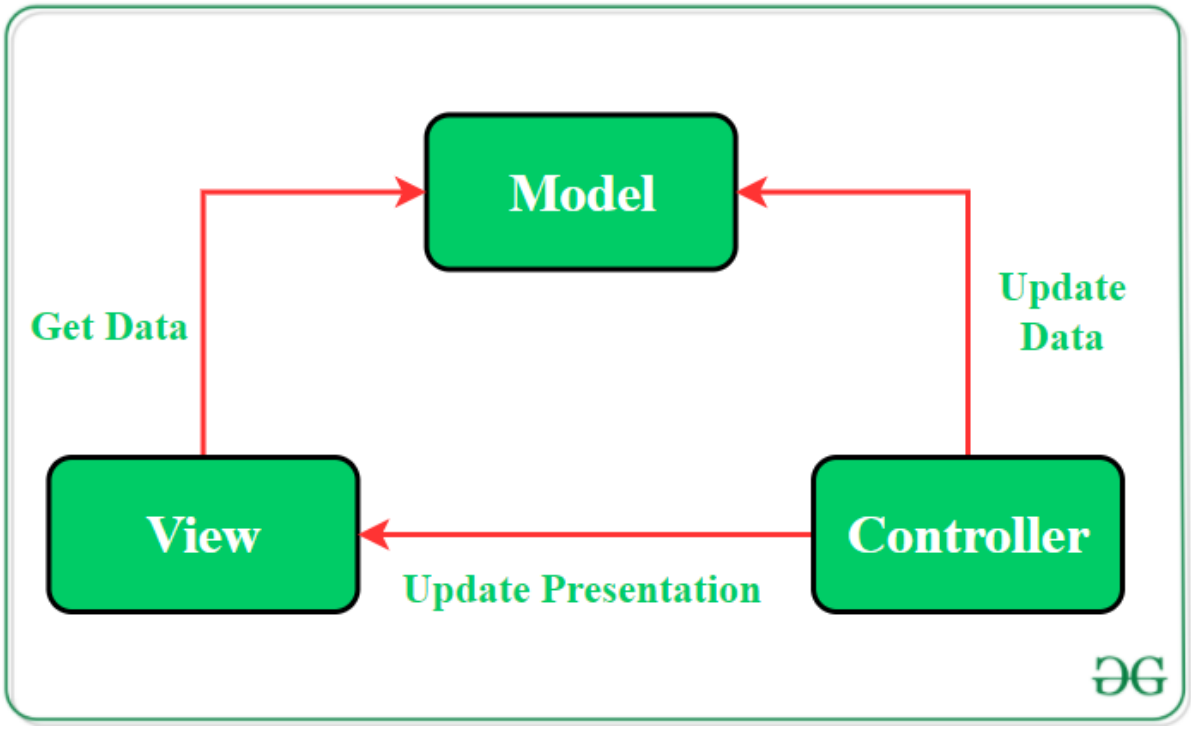

Рис.2.1. Шаблон Модель-Подання-Контролер (MVC)

Паттерн MVC (рис.2.1) пропонує розбити код на 3 компоненти. При створенні класу/файлу програми розробник повинен віднести його до одного з наступних трьох рівнів:

Модель: цей компонент зберігає дані програми. Він не має жодних знань про інтерфейс. Модель відповідає за обробку логіки предметної області (реальні бізнес-правила) та зв'язок з рівнями бази даних та мережі.

Перегляд: це рівень інтерфейсу користувача (користувацький інтерфейс), який містить компоненти, видимі на екрані. Більш того, він забезпечує візуалізацію даних, що зберігаються в моделі, та пропонує взаємодію з користувачем.

Контролер: цей компонент встановлює зв'язок між поданням та моделлю. Він містить основну логіку програми, отримує інформацію про реакцію користувача та оновлює модель відповідно до потреб.

Шаблон Модель-Подання-Презентатор (MVP)

Шаблон MVP(рис.2.2) вирішує проблеми MVC та забезпечує простий спосіб структурування кодів проектів. Причина широкого визнання MVP полягає в тому, що він забезпечує модульність, можливість тестування, а також чистішу та підтримувану кодову базу. Він складається з наступних трьох компонентів:

Модель: Шар зберігання даних. Він відповідає за обробку логіки предметної області (реальні бізнес-правила) та зв'язок з рівнями бази даних та мережі.

Вигляд: рівень інтерфейсу користувача (інтерфейс користувача). Він забезпечує візуалізацію даних та відстеження дій користувача з метою сповіщення Presenter.

Ведучий: витягує дані з моделі і застосовує логіку інтерфейсу користувача, щоб вирішити, що відображати. Він керує станом подання та робить дії відповідно до повідомлення користувача про введення з подання.

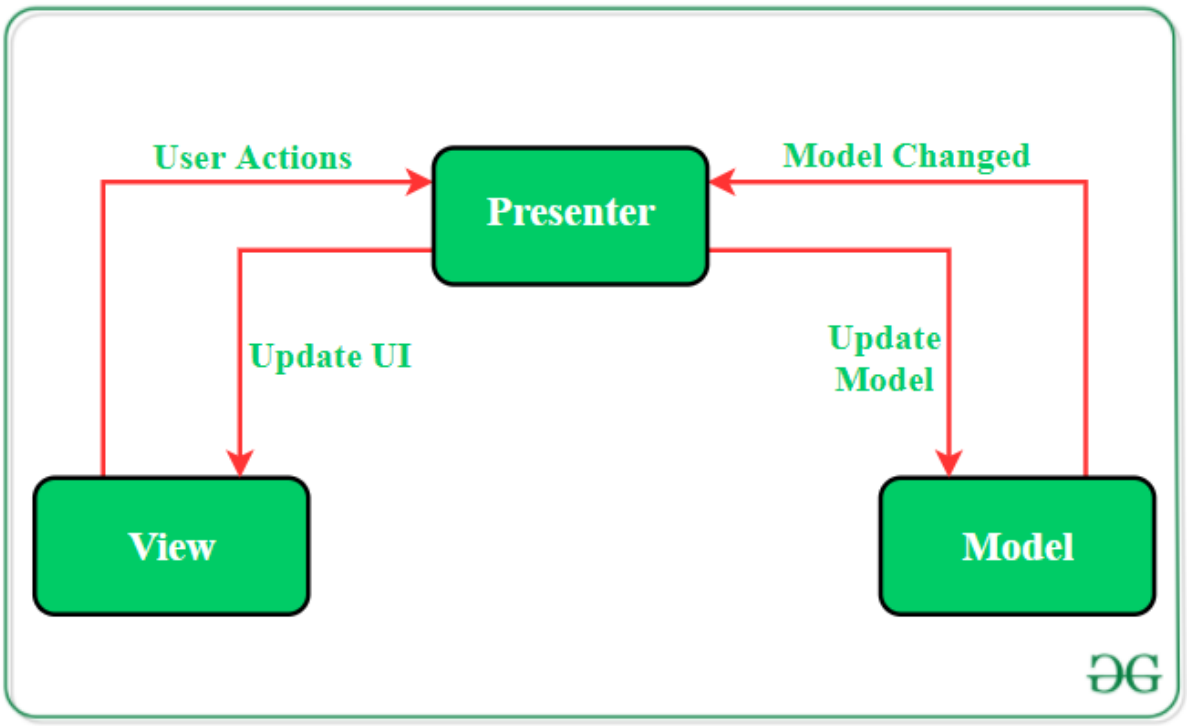

Рис.2.2. Шаблон Модель-Подання-Презентатор (MVP)

Шаблон Модель - Подання - ViewModel (MVVM)

Шаблон MVVM(рис.2.3) має деяку подібність [15] із шаблоном проектування MVP (Model - View - Presenter), оскільки роль Presenter грає ViewModel. Однак недоліки шаблону MVP були усунені за допомогою MVVM. Він пропонує відокремити логіку подання даних (уявлення або інтерфейс користувача) від основної частини бізнес-логіки програми. Окремі рівні коду MVVM:

Модель: цей рівень відповідає за абстракцію джерел даних. Модель та ViewModel працюють разом, щоб отримати та зберегти дані.

Подання. Мета цього рівня – інформувати ViewModel про дії користувача. Цей рівень спостерігає за ViewModel і не містить жодної логіки програми.

ViewModel: надає ті потоки даних, які стосуються представлення. Більше того, він служить сполучною ланкою між Моделью та Поданням.

Ключова відмінність між MVP і MVC полягає в тому, що презентатор MVP відіграє більш активну роль у взаємодії між моделлю і поданням і відповідає за управління потоком даних між ними. MVVM означає Модель-

Подання-ViewModel. У шаблоні MVVM «Модель» представляє дані [15] та логіку програми, «Уявлення» представляє інтерфейс програми користувача, а «ViewModel» — це рівень, який знаходиться між моделлю і поданням і відповідає за представлення дані та логіку моделі в подання таким чином, щоб з ними було легше працювати. ViewModel також обробляє введення користувача та оновлює модель за необхідності.

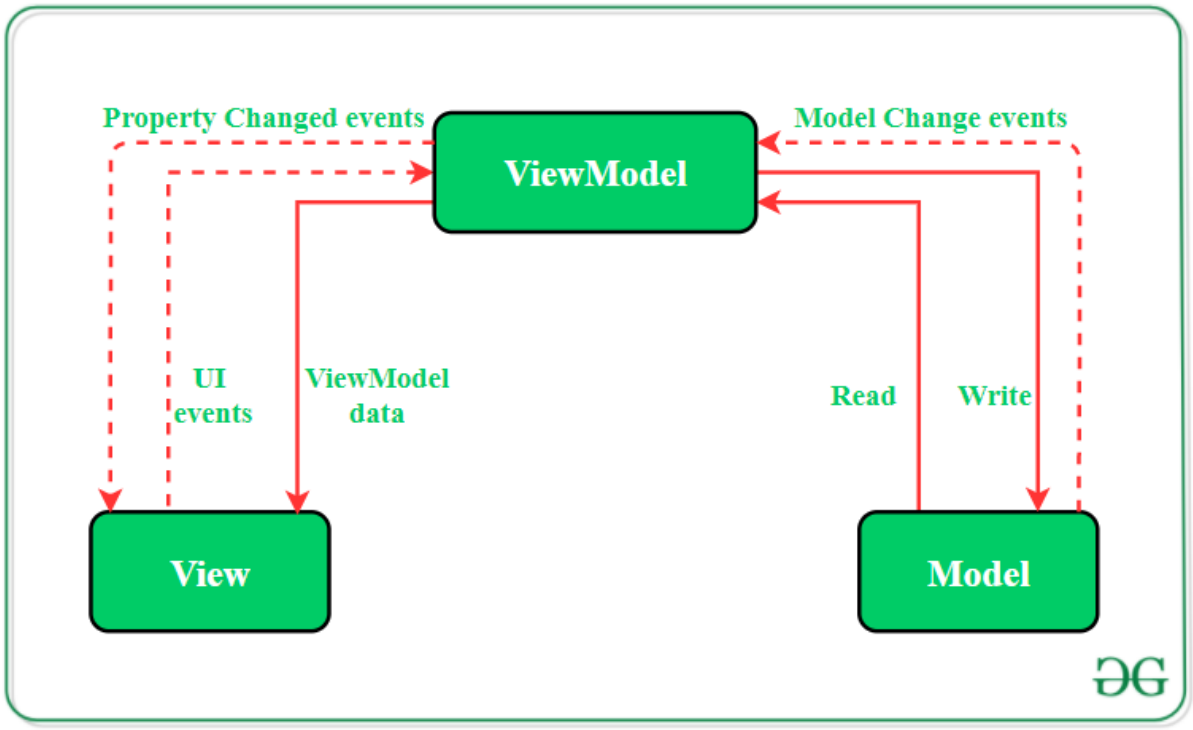

Рис.2.3. Шаблон Модель - Подання - ViewModel (MVVM)

У цілому нині, основна різниця між цими патернами залежить від ролі медіаторного компонента. І MVC, і MVP включають контролер або презентатор, який діє як посередник між моделлю і поданням, тоді як MVVM включає модель представлення, яка служить посередником між моделлю і поданням. MVC — найпростіший із цих шаблонів, тоді як MVP та MVVM більш гнучкі та дозволяють чіткіше розділити завдання між різними рівнями програми.

Одже для реалізації додатку буде доречним обрати модель MVP оскільки Архітектура MVP (Model-View-Presenter) є однією з найпопулярніших архітектур для створення додатків на Flutter. Вона ґрунтується на принципі

розділення відповідальності, що дозволяє розробникам створювати більш організований, зрозумілий, тестований, модульний і ефективний код.

Шар моделі відповідає за зберігання даних і бізнес-логіку. Він не знає про відображення і не повинен знати, як воно реалізовано. Це дозволяє розробникам тестувати модель незалежно від відображення.

Шар відображення відповідає за відображення даних на екрані. Він не знає про модель і не повинен знати, як вона працює. Це дозволяє розробникам тестувати відображення незалежно від моделі.

Шар Presenter відповідає за зв'язок між моделлю та відображенням. Він отримує дані від моделі і передає їх відображенню. Він також отримує відображення і передає його моделі. Це дозволяє розробникам зосередитися на взаємодії між моделлю та відображенням, не вдаючись до деталей реалізації кожного з них.

Переваги архітектури MVP

Розділення відповідальності: MVP дозволяє розробникам розділити відповідальність за додаток між трьома шарами, що полегшує розуміння та обслуговування коду.

Легкість тестування: MVP легко тестується, оскільки кожен шар може бути протестований окремо.

Модульність: MVP дозволяє легко додавати нові функції до додатку, оскільки кожна функція може бути реалізована в окремому шарі.

Ефективність: MVP може бути ефективним у плані продуктивності, оскільки кожен шар може бути оптимізований для виконання своєї конкретної задачі.

Причини вибору архітектури MVP для створення додатків на Flutter

Flutter є фреймворком, який орієнтований на відображення. MVP є ідеальною архітектурою для цього типу фреймворків, оскільки вона дозволяє розділити відповідальність за відображення від відповідальності за бізнес-логіку.

Flutter є фреймворком, який підтримує гнучке програмування. MVP добре працює з реактивним програмуванням, оскільки воно дозволяє легко зв'язувати шар відображення з шаром моделі.

Flutter є фреймворком, який підтримує модульність. MVP дозволяє легко створювати модульні додатки, що є важливою перевагою для Flutter.

Висновок

Архітектура MVP є хорошим вибором для створення додатків на Flutter, оскільки вона має ряд переваг, включаючи розділення відповідальності, легкість тестування, модульність і ефективність. Однак вона може бути складною для розуміння і реалізації, а також не дуже ефективною для невеликих додатків.

Основний вид взаємодії елементів у створеному додатку має вид(рис.2.4):

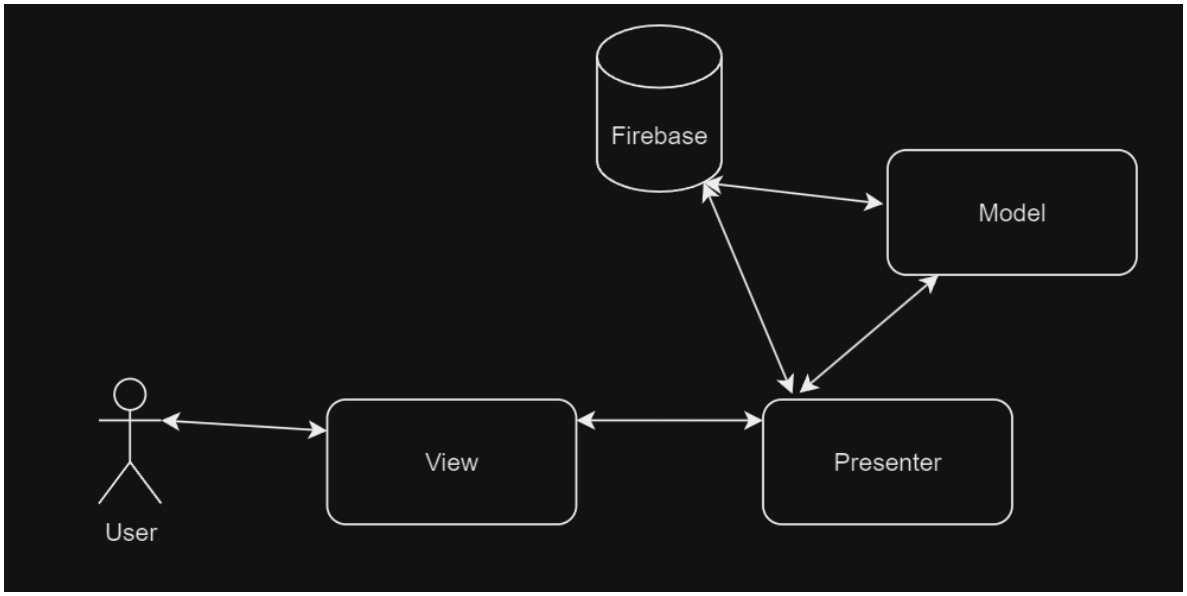

Рис.2.4. Зображення структури додатку

Користувач взаємодіє із інтерфейсом додатку – Представленням, передаючи дані між класами та програмними компонентами що їх обробляють чи зберігають. Після обробки даних змінюється стан представлення, з’являються нові елементи інтерфейсу або змінюються поточні.

2.2. Основні програмні складові додатку та їх реалізація

2.2.1. Реєстрація у додатку

Для реєстрації у додатку було створено 4 варіанти зав'язані на інструментах які впроваджує Firebase – Firebase Auth, та також на методах пов'язаних із конкретним варіантом реєстрації.

Для впровадження реєстрації із допомогою Firebase треба:

1) Додати додаток у Firebase та налаштувати його(рис.2.5):

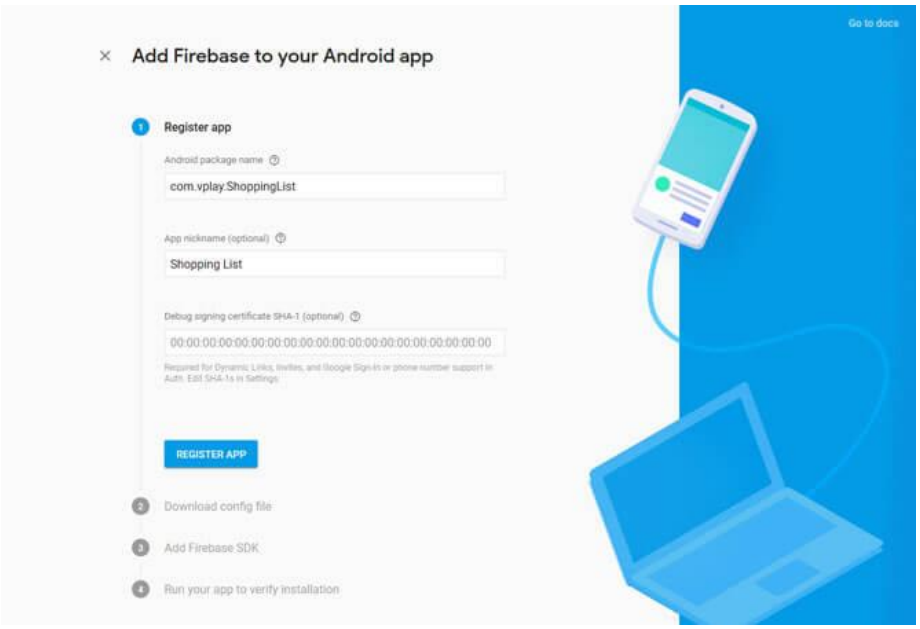

Рис.2.5. Початкове налаштування проекту у Firebase

2) Після перших кроків налаштування завантажити файл google-services.json, та додати файл у проект додатку. Цей файл тримає у собі основну інформацію про зареєстрований проект у Firebase та має вигляд:

```
{
  "project_info": {
    "project_number": "766025596428",
    "firebase_url": "https://shoppinglist-vplay.firebaseio.com",
    "project_id": "shoppinglist-vplay",
    "storage_bucket": "shoppinglist-vplay.appspot.com"
  },
  "client": [
```

```
    {
      "client_info": {
        "mobilesdk_app_id": "1:766000006428:android:a28d200000001c88",
        "android_client_info": {
          "package_name": "com.vplay.ShoppingList"
        }
      },
      "api_key": [
        {
          "current_key": "AIzaSyCvAzxxxxxxxxxxxxxxxxEXg-cv8TikY0"
        }
      ]
    }
  ],
  "configuration_version": "1"
}
```


З його допомогою та додавши необхідні бібліотеки у файли додатку ми зможемо створити механізм авторизації користувачів у додатку, чиї дані будуть знаходитися у Firebase після створення облікового запису.

Або прискорити процес конфігурації додатку та налагодження зв’язку можна зробити за допомогою інструментарію FlutterFlow(рис.2.6):

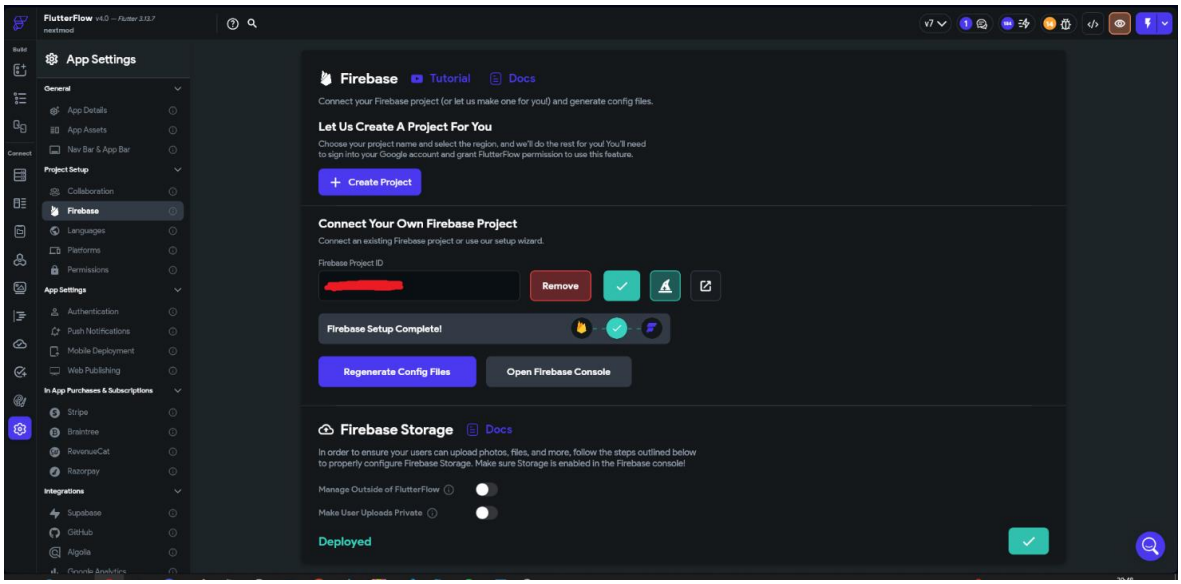

Рис.2.6. Підключення Firebase до FlutterFlow

На малюнку зображений розділ, у якому можна зв’язати базу даних із проектом додатоку, цей сервіс використовуючі власний інструментарій

виконає налагодження автоматично. Після успішного підключення бази даних необхідно налагодити аутентифікацію, це буде зроблено також через функціонал даного сервісу, котрий надає можливість створити аутентифікацію за допомогою особистих записів Google, Facebook, Apple та особисту пошту. Відповідні налаштування необхідно зробити і у Firebase(рис.2.7):

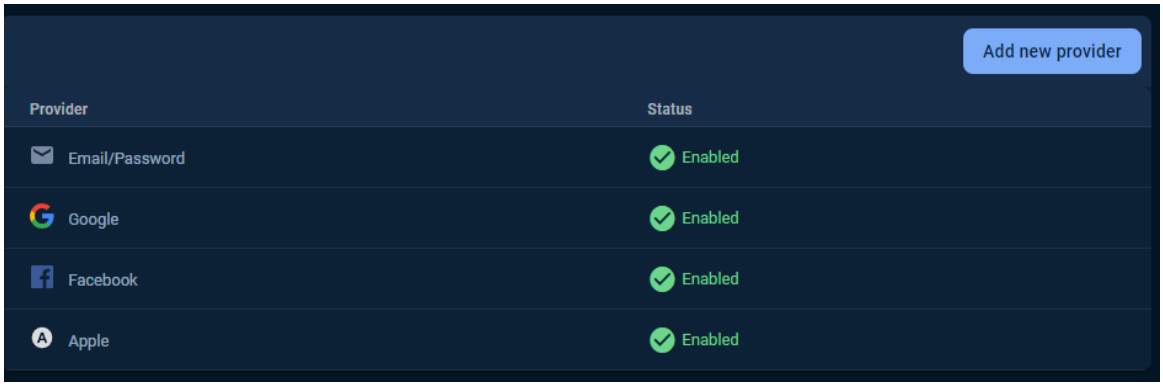

Рис.2.7. Перелік підключених сервісів

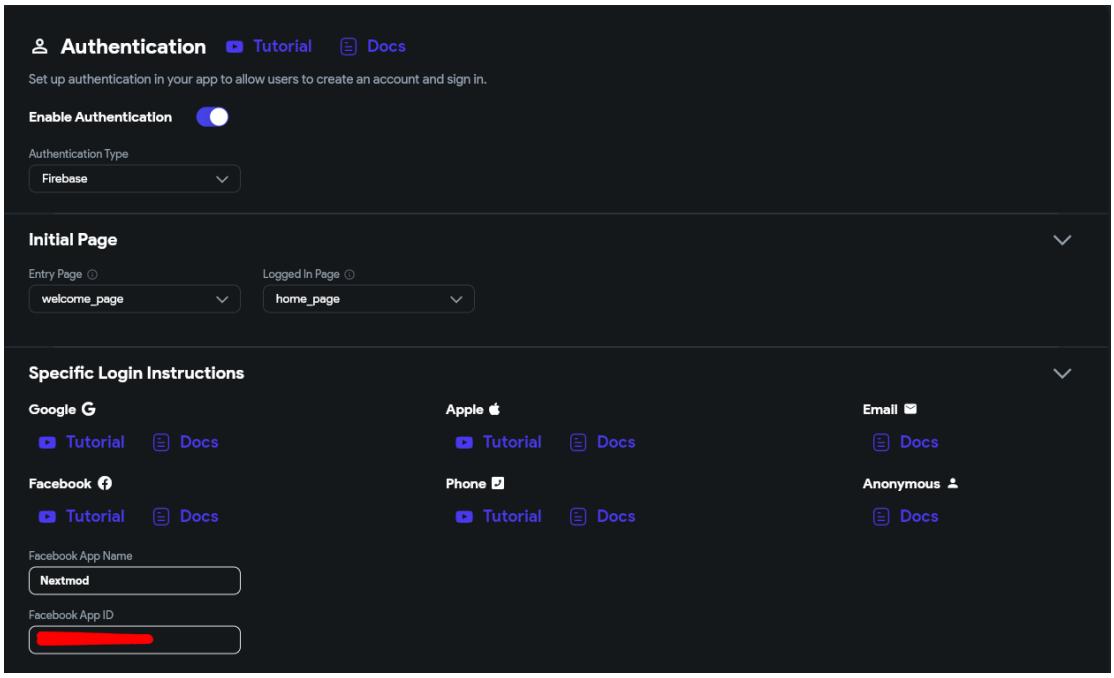

Рис.2.8. Сторінка налагодження авторизації

На сторінці налагодження авторизації(рис.2.8) обираємо початкову сторінку, на котрій будуть представлені варіанти авторизації у додатку та домашню сторінку, кота буде відображатися кожного разу при запуску додатку якщо користувач вже авторизований.

Після переносу коду з FlutterFlow ми побачемо що були створені файли(рис.2.9) у проекті у яких кожен відповідає за свій спосіб авторизації користувача:

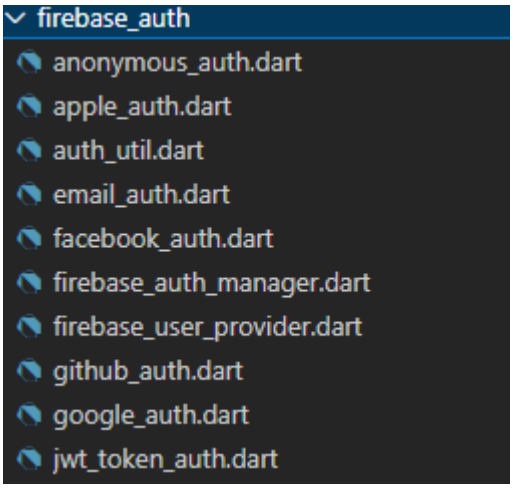

Рис.2.9. Перелік класів структур даних

Макет(рис.2.10) послідовності Авторизації користувача у додатку:

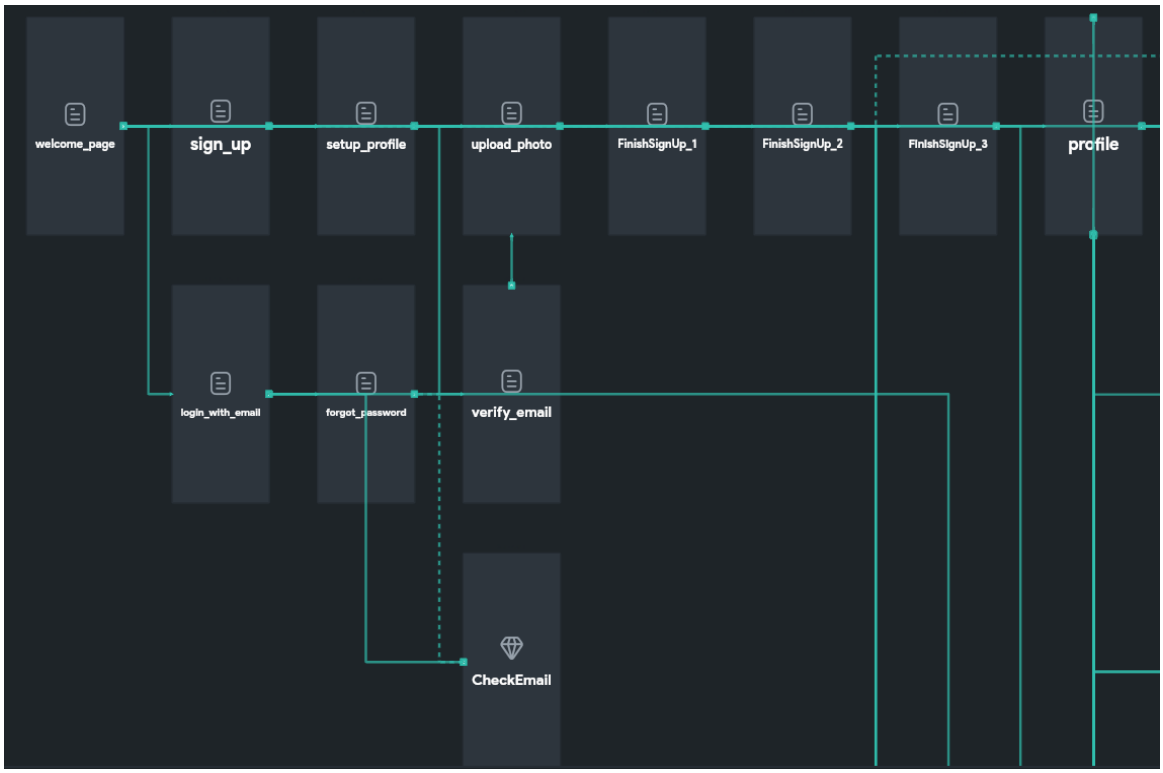

Рис.2.10 Послідовність сторінок при авторизації

У першу чергу після першого входу у додаток користувача зустрічає сторінка привітання із варіантами авторизації у додатку. Тут користувач може брати найбільш оптимальний для нього спосіб реєстрація через пошту,

Facebook, Apple або Google. Усі ці способи зав’язані на функціоналі що впроваджує Firebase, тому механізм в них схожий. У випадках, коли користувч обирає спосіб не пов’язаний із його особистою поштою викликається ділогове вікно пов’язане із певним сервісом(рис.2.11):

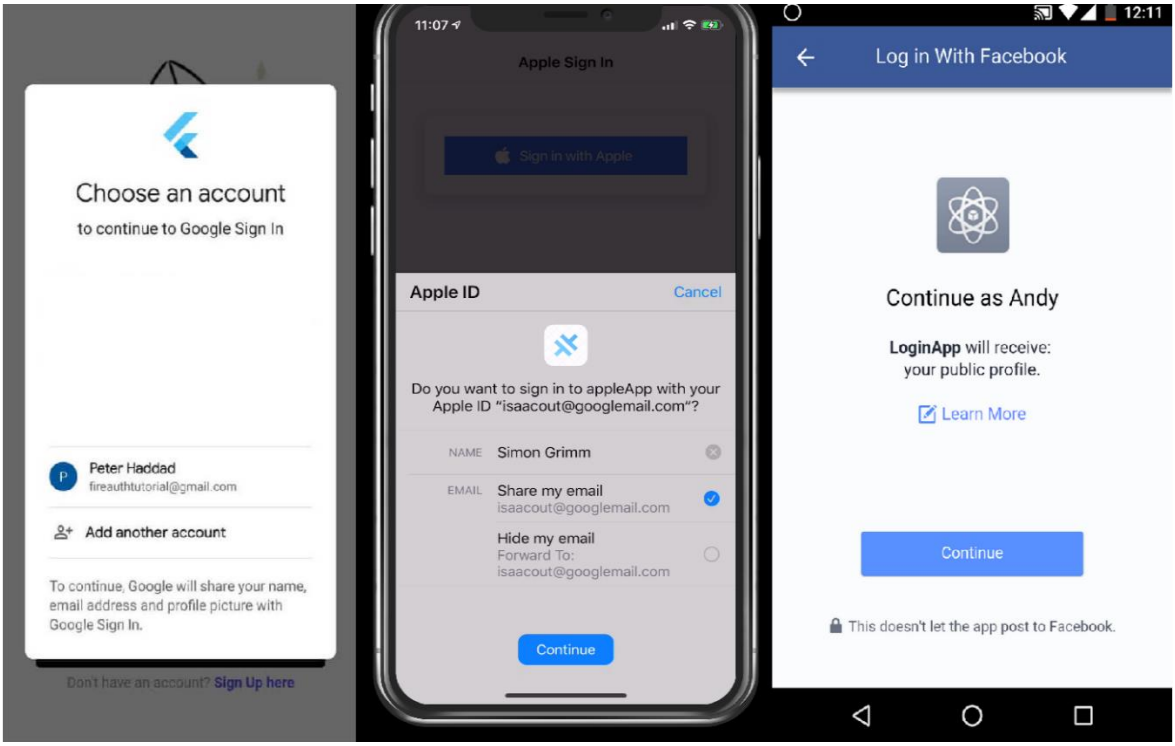

Рис.2.11. Мудульні вікна авторизації від різних сервісів

Роздивимося авторизацію за допомогою сервісів на прикладі(рис.2.12) Google Auth [16].

Рис. 2.12. Представлення взаємодії між сервісами для авторизації

Рис.2.13. Документ користувача у базі даних

Після успішної авторизації створюється запис користувача(рис.2.13) у базі даних що має вигляд.

При реєстрації створюється документ користувача у базі даних, що містить такі початкові поля: дата реєстрації, імя, пошта, Інстаграм, місцеположення, путь до зображення аватару та інш.

2.2.2. Створення запису об’яви з ТЗ та запчастинами та розміщення даних у додатку та структури бази даних

Основними сутностями у додатку є 3 Сутності: Користувач, Тоу та Mod. Тоу представляє собою транспортний засіб із своїми особливостями, котрий створюють користувачі додатку, розміщуючи їх у себе в профілі. Mod – це дочірня сутність ТЗ що представляє собою запчастину для створеного транспортного засобу, окремо вона не може існувати без батьківської сутності.

Оскільки розробка додатку починалась у FlutterFlow, не було можливості створювати звичні декілька слоїв підколекції, для того щоб у кожного

користувача була підколекція ТЗ, а в ТЗ своя підколекція запчастин приклад(рис.2.14):

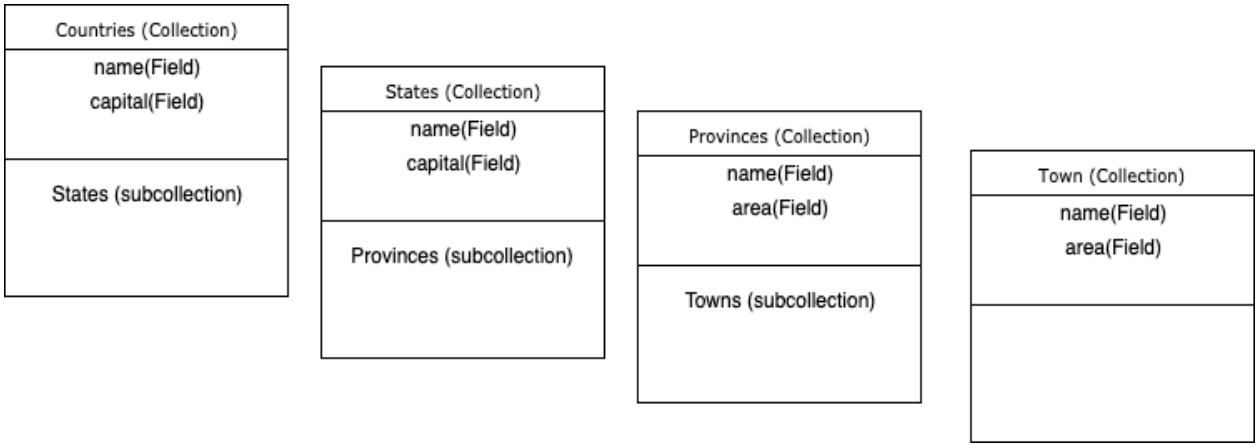

Рис.2.14. Приклад ієрархії БД із підколекціями

На прикладі зображений зв’язок сутностей у якій кожний наступний елемент є дочірнім для попереднього. Тому було прийнято рішення створити двусторонній зв’язок між усіма основними сутностями, що має вигляд(рис.2.15):

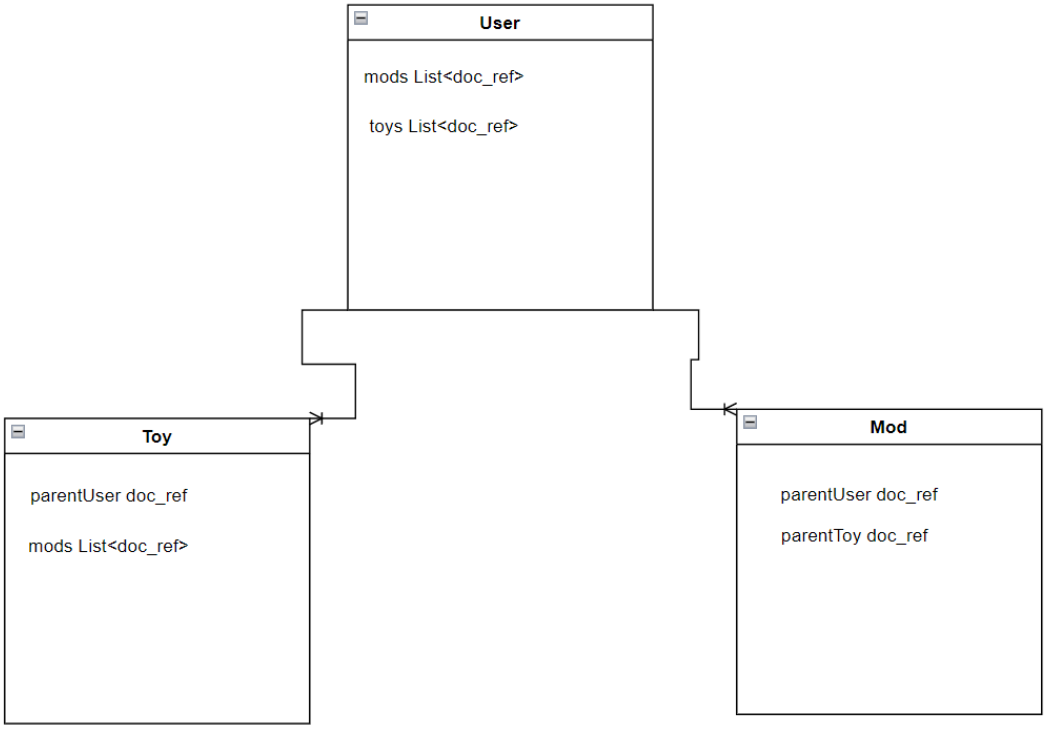

Рис.2.15. Зв’язок між сутностями у БД

Структура бази даних матиме вид(рис.2.16):

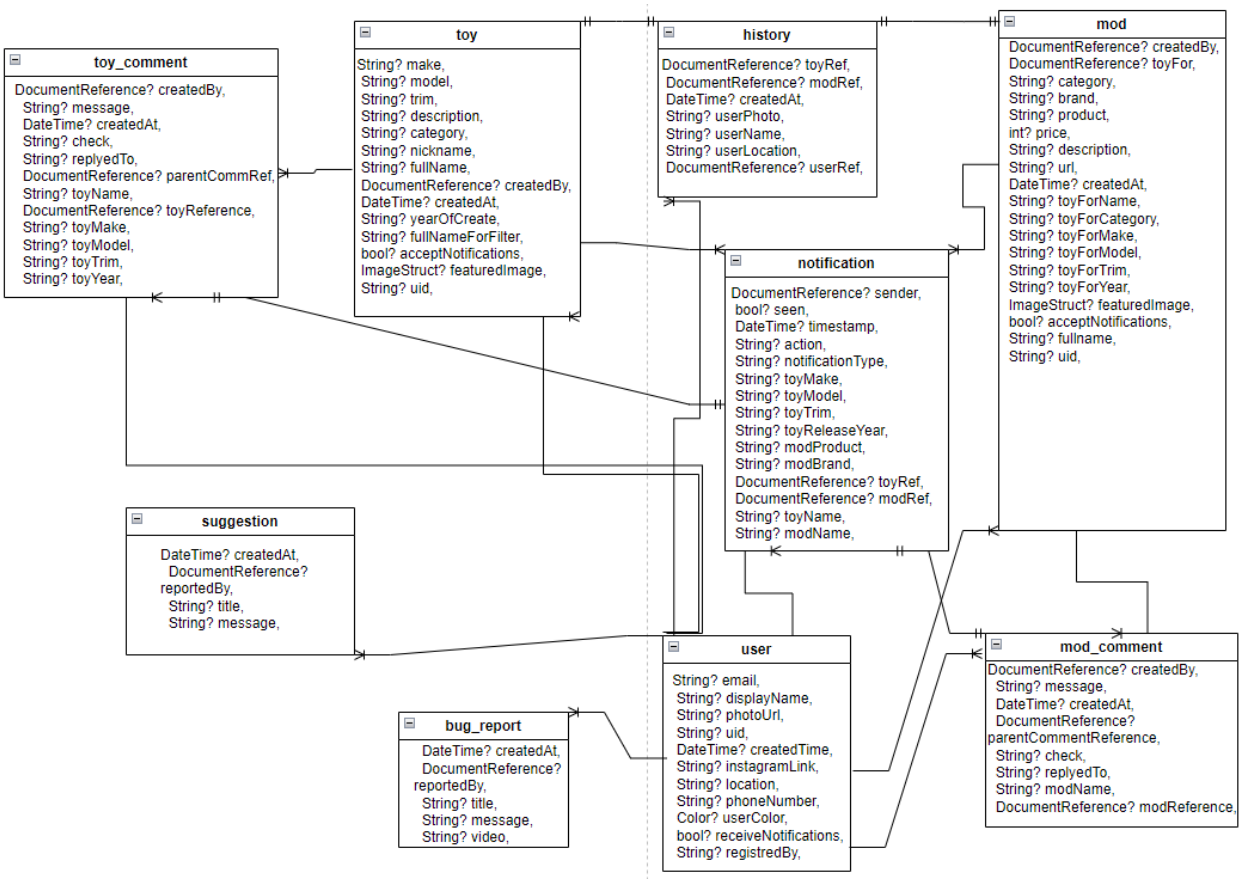

Рис.2.16 Структура БД додатку

Запис користувача має списки посилань на документи Тоїв та Модів у базі даних, що знаходяться у своїх колекціях, котрі у свою чергу мають посилання на своїх батьків. Для Тою батьківською сутністю є Користувач, із списком посилань на всі Тої котрі йому належать, а дочірніми сутностями є Моди, список посилань на корті є у документі Тою. Що до сутності Моду, він містить у собі два посилання: на користувача що його створив, та на Той до котрого він відноситься.

Створення запису Тоя складається із декількох етапів:

1) Вибір року виготовлення, бренду, моделі, категорії та комплектації.
2) Опис ТЗ та додавання йому «імені».
3) Завантаження зображень до нього та фотографій.
4) Підтвердження створення та створення запису у базі даних.

Однак, перед заповненням даних, створюється структура даних для Тоїв що матиме вид:

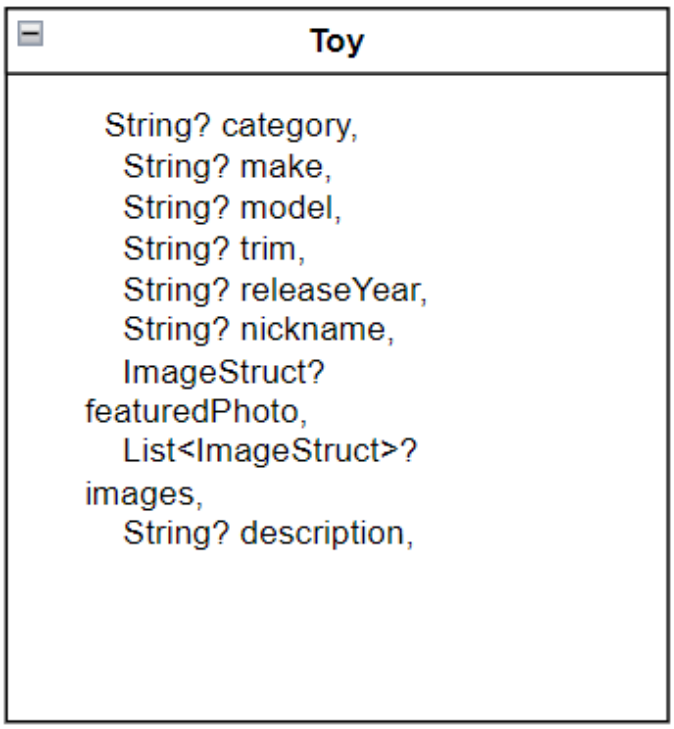

Рис.2.17. Структура даних ТЗ перед завершенням створення

На зображенні(рис.2.17) бачимо усі дані що будуть заповнені під час створення Тою, однак не всі з них необхідні.

Після заповнення усіх необхідних даних для створення тою, створюється запит до бази даних для створення запису документу у відповідній колекції, структура документу що буде знаходитися у базі даних суттєво відрізняється від тієї що використовувалась для заповнення даних, та матиме вид(рис.2.18):

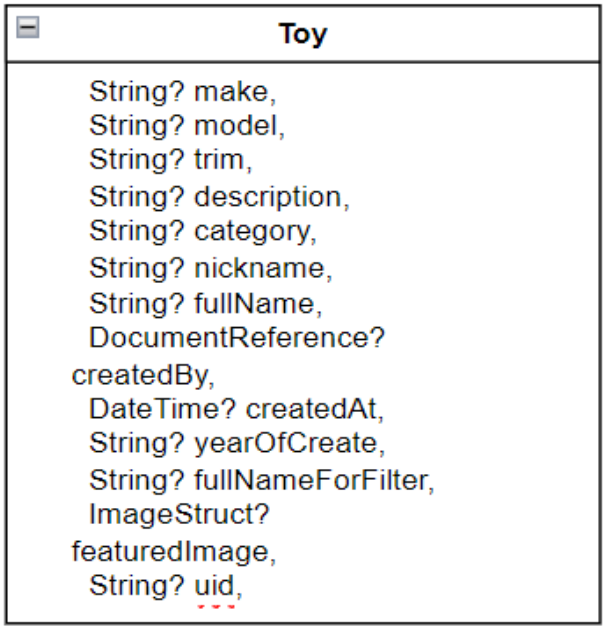

Рис.2.18. Структура документу ТЗ після створення

Додані поля, що будуть у майбутньому використані для фільтрації ТЗ у додатку відповідно запитам користувачів. Також додане посилання на Користувача(рис.2.19), котрий створив даний запис ТЗ, для створення зв'язку між цими двома сутностями.

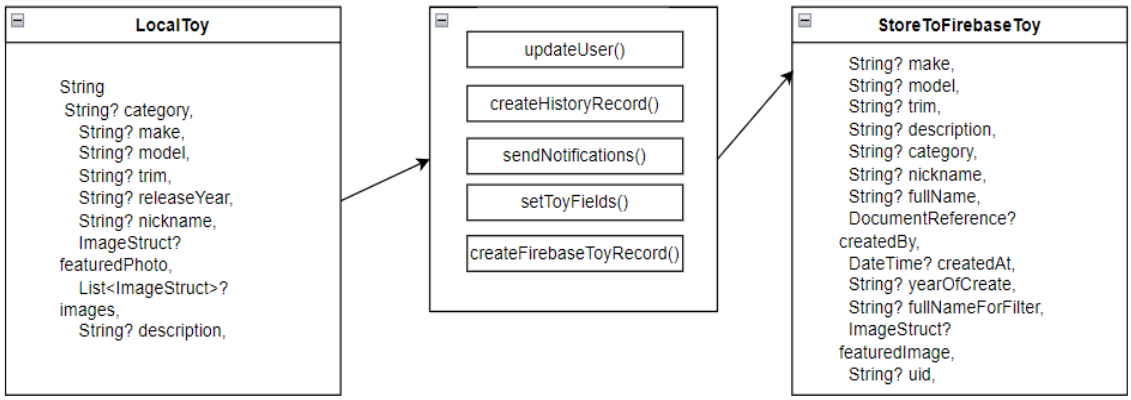

Рис.2.19. Процес заповнення даними

Після успішного створення запису Тоя у базі даних, усі користувачі додатку, котрі відстежують Користувача, котрий створив даний запис ТЗ, отримають Push-повідомлення на їхні мобільні пристрої, а додаток перенесе користувача до його профіля де він побачить створений ТЗ запис. Також буде виконано створення запису для відображення створеного ТЗ або деталі у загальному списку що бачать усі користувачі – Історії(рис.2.20).

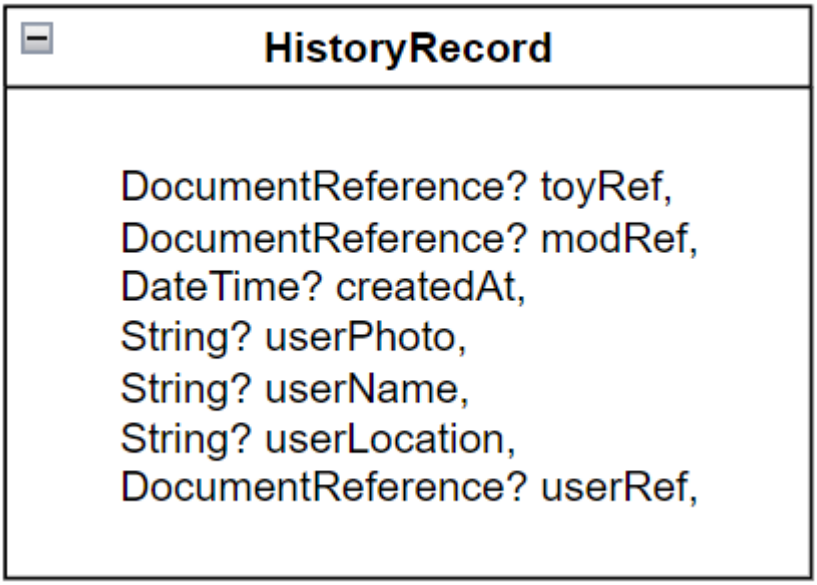

Рис.2.20. Структура даних компоненту Історії

Історія необхідна для просування об’яв створених користувачами, щоб інші користувачів бачили нещодавно створені записи із ТЗ та запчастинами, що значно полегшує взаємодію між користувачами додатку.

Для розміщення даних що необхідно отримати з бази даних та відобразити у стовпчиках або списках(рис.2.21) відображення, як на Рис. використовується віджет StreamBuilder.

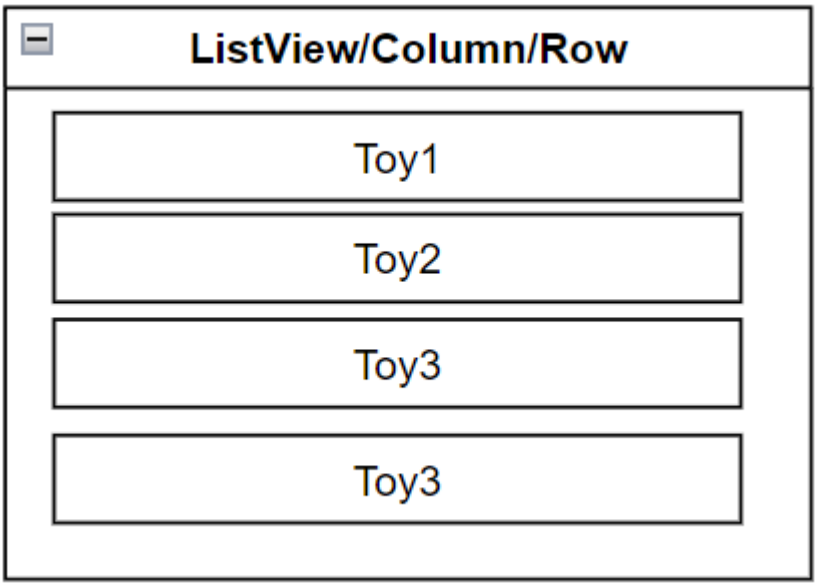

Рис.2.21. Представлення списків

StreamBuilder - це клас виджета Flutter, який використовується для відображення даних, які надходять із потоку. Потік - це джерело даних, яке може генерувати нові значення з часом. StreamBuilder використовує метод builder(), щоб повторно створити свій вміст при зміні потоку. Цей інструмент доцільно буде використовувати для отримання документів з бази даних, та відстежувати зміни у пов’язаній із віджетом колекції.

```
child: StreamBuilder<List<HistoryNewRecord>>(
  stream: queryHistoryNewRecord(
    queryBuilder: (historyNewRecord) => historyNewRecord
        .where('user_ref', isNotEqualTo: currentUserReference),
  ),
```

Рис.2.21. Приклад коду віджета StreamBuilder

На(рис.2.21) зображено приклад коду реалізації StreamBuilder, котрий отримує колекцію із записами створених об’яв, та фільтрує так щоб поточний

користувач бачив тільки записи створені іншими користувачами. При зміні даних у потоці виконується метод builder для повторної генерації вмісту StreamBuilder.

2.2.3. Фільтрація створених ТЗ та запчастин

Однією із найважливіших функцій будь-якого додатку чи сервісу із великою кількістю записів Вона дозволяє користувачам обмежити список об'яв за певними параметрами, такими як: марка та модель транспортного засобу, рік випуску, комплектація тощо.

Покупець, який шукає новий автомобіль, може використовувати фільтрацію, щоб обмежити список об'яв лише автомобілями, які були випущені в поточному році. Користувач, який шукає запчастину для свого автомобіля, може використовувати фільтрацію, щоб обмежити список об'яв лише запчастинами для певної марки та моделі автомобіля.

Сутність Тоу має такі основні поля: Category, Year of Create, Make, Model, Trim, тому доцільно буде створити такий алгоритм фільтрації із функціоналом, котрий дозволить користувачу фільтрувати об’яви саме за цими параметрами. Також, для більшої зручності та покращеного досвіду користування додатком, буде розроблений алгоритм що дозволить користувачу обирати тільки пов’язані між собою дані. Наприклад ми маємо 20 ТЗ різних категорій, марок та моделей, користувач обирає категорію ТЗ автомобілі, тому відповідно для цієї категорії будуть представлені лише моделі, марки та комплектації які відносяться до категорії автомобілів. Ієрархія залежностей між сутностями виглядатиме так(рис.2.22):

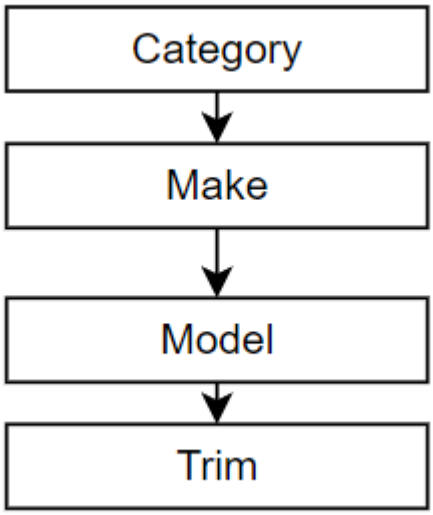

Рис.2.22. Ієрархія сутностей

Якщо користувач, обрав певну сутність, а після обрав сутність вище за ієрархією, то усі поля нижче будуть очищені, для того щоб запобігти відображення даних що не відносяться до відфільтрованих ТЗ за сутністю що знаходиться вище за ієрархією. Виглядатиме це так(рис.2.23):

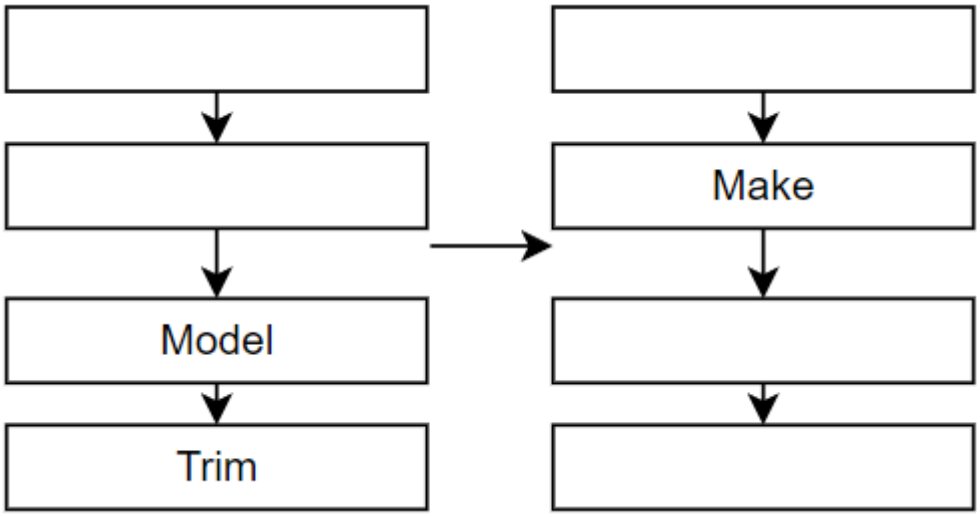

Рис.2.23. Результат зміни параметру вище за ієрархією

Спочатку користувач обрав Модель та Комплектацію перед цим не обравши Марку та категорію ТЗ, якщо він обере варіант фільтрації за сутністю вище поточної найвищої ступені ієрархії, то дані що є дочірніми будуть очищені, та заміняться на дані що є дочірніми відносно обраної сутності.

Для фільтрації даних, спочатку необхідно їх отримати та розташувати у пам’яті пристрою, для цього проходить звернення до бази даних, колекції

“toys”. Після звернення заповнюються 4 списки із документами ТЗ, усі вони мають в собі однакові дані, але їх призначення різне, для кожного параметру фільтрації.

```
if (FFAppState().filterReceivedToyCategory.isEmpty) {
  var snapshot =
      await FirebaseFirestore.instance.collection("toys").get();

  FFAppState().filterReceivedToyCategory = snapshot.docs
      .map((e) => ToysRecord.getDocumentFromData(e.data(), e.reference))
      .toList();

  FFAppState().filterReceivedToyMake = snapshot.docs
      .map((e) => ToysRecord.getDocumentFromData(e.data(), e.reference))
      .toList();

  FFAppState().filterReceivedToyModel = snapshot.docs
      .map((e) => ToysRecord.getDocumentFromData(e.data(), e.reference))
      .toList();
  FFAppState().filterReceivedToyTrim = snapshot.docs
      .map((e) => ToysRecord.getDocumentFromData(e.data(), e.reference))
      .toList();
}
```

Рис.2.24. Заповнення списків із даними для фільтрації

Усі створені 4 списки(рис.2.24) із даними будуть оброблятися методом deleteNonEquals(String value), що приймає назву певної сутності для фільтрації. Цей метод різниться для кожного варіанту фільтрації, чим вище ієрархія сутності за котрою фільтрують – тим більше списків необхідно редагувати. Якщо користувач фільтрує за Категоріями, то редагуються списки ТЗ для Марок, Моделей та Комплектацій. Для фільтрації ТЗ за категоріями, виконання цього методу буде виглядати так(рис.2.25):

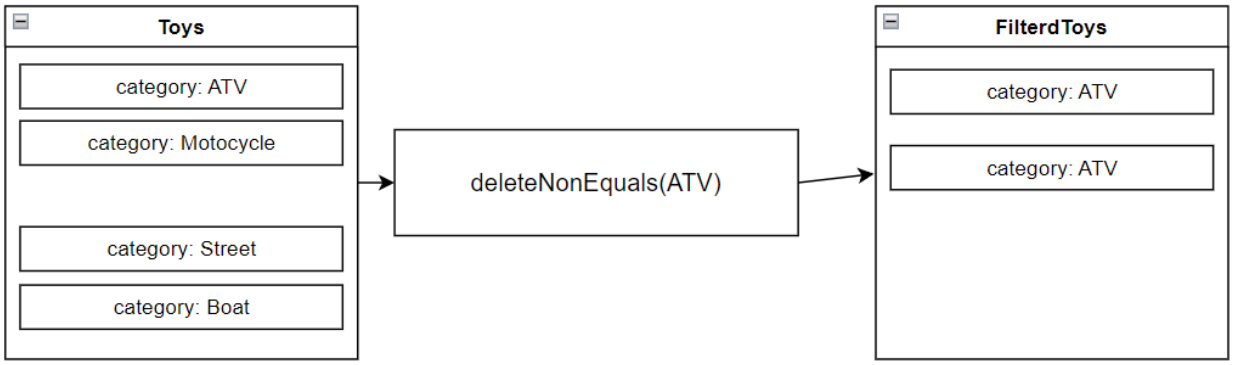

Рис.2.25. Зображення виконання методу

У кожному списку нижче за ієрархією видаляються документи, у котрих назва Категорії не дорівнюється значенню що передано до методу що оброблює списки. Приклад коду фільтрації за категоріями(рис.2.26):

```
Future deleteNonEquals(String category) async {
  FFAppState()
      .filterReceivedToyMake
      .removeWhere((element) => element.category != category);

  FFAppState()
      .filterReceivedToyModel
      .removeWhere((element) => element.category != category);

  FFAppState()
      .filterReceivedToyTrim
      .removeWhere((element) => element.category != category);
}
```

Рис.2.26. Приклад коду методу

На виході ми матимемо 3 відфільтровані списки із ТЗ у котрих категорія буде дорівнювати значенню категорії за котрою відбувається фільтрація.

Завжди незмінним під час фільтрації буде залишатися лише список пов’язаний із Категоріями, адже він знаходиться у верхівці ієрархії, в ньому буде знаходитися усі ТЗ нефільтровані, як батьківський список. Оскільки дані у списках нижче за ієрархією будуть завжди змінюватись, є необхідність отримувати початково не змінені дані для випадків, коли користувач буде очищувати дані для фільтрації, тому у таких випадках усі списки за ієрархією нижче при очищенні будуть перезаписувати у себе дані із списків за ієрархією вище (Рис.2.27):

Category	Category	Category	Category
default	default	default	default
Make	**Make**	**Make**	**Make**
default	by category	default	default
Model	**Model**	**Model**	**Model**
default	by category	by make	default
Trim	**Trim**	**Trim**	**Trim**
default	by category	by make	by model

Рис.2.27. Приклад зміни параметрів

```
setState(() {
  FFAppState()
          .filterReceivedToyTrim =
      List.from(FFAppState()
          .filterReceivedToyMake); // List.from
  FFAppState()
      .toyTrimForFilter = '';

  FFAppState()
          .filterReceivedToyModel =
      List.from(FFAppState()
          .filterReceivedToyMake); // List.from
  FFAppState()
      .toyModelForFilter = '';
});
```

Рис.2.28. Приклад коду з перезаписом даних

Цей код(рис.2.28) буде замінювати дані із попередньо відсортованих списків на дані із списку вище за ієрархією, у даному випадку списки Комплектацій та Моделей на дані у списку Марок ТЗ.

Окремо від фільтрацій за вище переліченими параметрами, є ще параметр фільтрації за роком, або діапазоном років створення ТЗ. Цей параметр для фільтрації є необхідним для такого типу додатків, оскільки однакові марки та моделі ТЗ можуть мати різні дати виготовлення, а відповідно до року виготовлення може бути різний стан ТЗ. Вибірка(рис.2.29) років буде знаходитись між 1940 роком та поточним. На відміну від отримання даних про Категорії, моделі, марки та комплектації з бази даних, роки будуть братися додатком із списку що початково створений у базових файлах додатку.

```
List<String> generateYears() {
  int currentYear = DateTime.now().year;
  List<String> yearsList = List.generate(
    currentYear - 1940 + 1,
    (index) => (1940 + index).toString(),
  );

  return yearsList.reversed.toList();
}
```

Рис.2.29. Приклад коду заповнення списку років

Для користувача буде доступно два випадаючих списки із роками: стартовий рік та кінцевий. Перший буде задавати ніжній поріг для пошуку по рокам, а другий, відповідно, крайню границю для пошуку між роками наприклад початковий рік 2001 та кінцевий 2018, звідци діапазон пошуку ТЗ буде мати вид 2001-2018 роки випуску. Якщо поля для років залишаться пустими, та користувач не обере початковий та кінцевий роки, то пошук буде проводиться у діапазоні усіх існуючих років. У випадку коли користувач обере один з параметрів, стартовий чи кінцевий, а другий залишиться не обраним, то для не обраного параметру буде прийнятий граничний рік для відповідного параметру.

Після формування даних запиту для фільтрації, коли користувач завершив обирати параметри за котрими буде проведена фільтрація, необхідно створити індивідуальний запит до бази даних в залежності від тих даних, що обрав користувач. Формування різних варіантів запиту до бази даних обумовлено політикою безпеки Firebase, що виключає формування запиту до БД із певними комбінаціями звернень, таких як декілька NOT-IN, IN, NON-EQUAL та інш. Тому буде доцільно розробити алгоритм, що визначатиме за якими даними необхідно шукати ТЗ. Фільтрація документів ТЗ проводиться по трьом параметрам: Категорія ТЗ, рік випуску ТЗ та ім’я ТЗ що сформоване із марки, моделі та комплектації ТЗ, якщо користувач не обрав роки випуску, то за замовчуванням завжди буде додаватися інтервал усіх років. Звідци, ми матимемо такі варіанти за котрими буде проводитися фільтрація:

1) Тільки за роками.
2) За категорією та роками.
3) За категорією та ім’ям.
4) За ім’ям та роками.

Для створення алгоритму створимо змінну флаг – “con”, що прийматиме дані перевірки параметрів та буде проводитись перевірка на не порожність параметрів(рис.2.30).

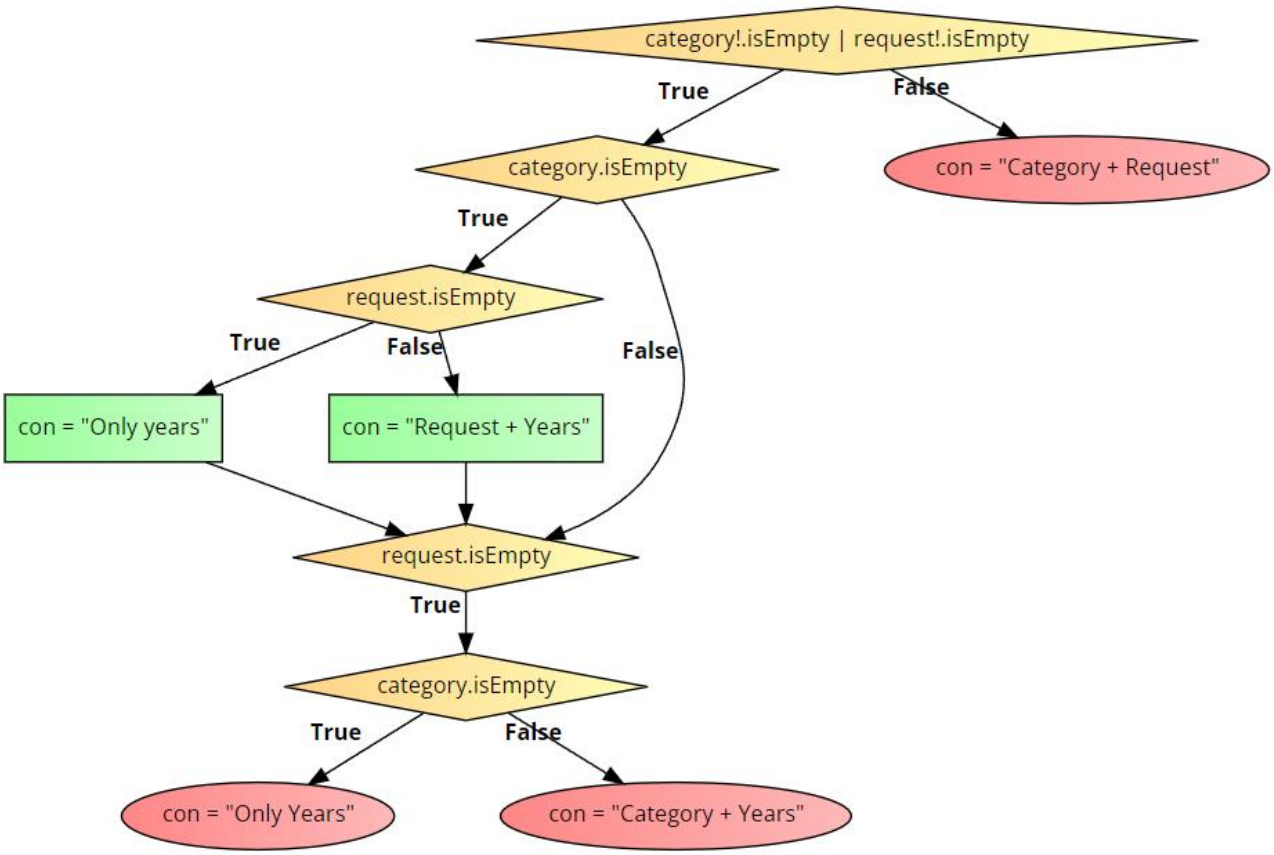

Рис.2.30. Блок-схема перевірки

Після перевірки параметрів, використовуючи switch-case, відповідно до типу запиту що зберігається у змінній “con”. Також необхідно не забувати про роки, котрих може бути більше ніж дозволяє Firebase, тому необхідно розбити масив років на кулька підмасивів або чанків. У цьому випадку максимальну кількість років у одному чанку буде дорівнювати 10, тому при початку фільтрації список років на 40 років, він буде розбитий на 4 списки по 10 років і для кожного буде проводитися фільтрація.

```
case "Request + Years":
  for (List<String> chunk in chunks) {
    QuerySnapshot snapshot = await FirebaseFirestore.instance
        .collection("toys")
        .where("year_of_create", whereIn: chunk)
        .get();
    snapshot.docs.forEach((doc) async {
      Map<String, dynamic> carData = doc.data() as Map<String, dynamic>;
      List<String> carName =
          carData['full_name'].toString().toLowerCase().split(' ');
      if (lowerRequest.every((element) => carName.contains(element))) {
        toys.add(doc.get('full_name'));
      }
    });
  }
  break;
```

Рис.2.31. Приклад коду одного із запитів

Цей код(рис.2.31) буде виконуватися у випадку якщо користувач обрав додав до параметрів фільтрації дані про Марку, Модель чи комплектацію та опціонально Роки. На початку створюється запит до бази даних колекції Toys, та отримує усі документи у котрих рік створення міститься у переданому чанку. Після отримання документів для кожного буде виповнюватися дії з формуванням у структуру даних Map<String,dynamic>, для отримання даних про ім’я ТЗ. Ім’я ТЗ початково є типом даних String, тому для пошуку у ньому відповідностей необхідно його розбити на складові по словам, та перевести у нижній регістр. Якщо у списку слів присутні усі слова із запиту, то цей документ буде додано у список із відфільтрованими ТЗ, що після будуть відображені у списку із знайденими ТЗ.

Зокрема фільтрації ТЗ у додатку має бути наявна і фільтрація запчастин усіх ТЗ. Основними параметрами, за котрими буде проходити фільтрація для запчастин – це їх категорія, та відношення до вже відфільтрваних ТЗ.

Документ запчастини у базі даних має в собі поля(рис.2.32), що дозволяють віднести при фільтрації запчастину до відфільтрованих ТЗ:

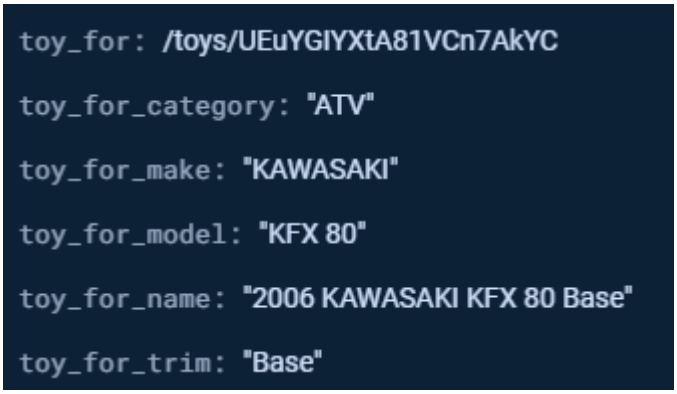

Рис.2.32. Параметри для фільтрації

По-перше це посилання на батьківський ТЗ, що використовується повсемісно у додатку для зв’язку ТЗ із дочірніми запчастинами.

По-друге це поля із Категорією, Моделлю, Маркою, Комплектацією та повною назвою ТЗ.

Після фільтрації ТЗ, також будуть відфільтровані і запчастини. З усього переліку запчастин залишаться лише ті, що є дочірніми для ТЗ що були

відфільтровані. Ця фільтрація відбувається за допомогою посилань на документи ТЗ у базі даних, вони отримуються після фільтрації та додаються у локальний список, до котрого додаток буде звертатися щоб отримати необхідні запчастини.

Алгоритм працює так, що якщо посилання на батьківський ТЗ запчастини міститься у відфільтрованому списку посилань на ТЗ, то він повертає цю запчастину та відображає у списку пов’язаних із ТЗ запчастинами.

До переваг даного створеного алгоритму фільтрації несумнінно можна віднести відсутність великої кількості звернень додатку до бази даних, що спростить роботу додатку, якщо користувач важлива швидкість отримання даних. Також простий алгоритм фільтрації є не вибагливим до обчислювальних ресурсів та швидким у виконанні.

РОЗДІЛ 3
РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ МОБІЛЬНОГО КРОСПЛАТФОРМНОГО ДОДАТКУ ІЗ ГНУЧКИМ АЛГОРИТМОМ ФІЛЬТРАЦІЇ ДЛЯ СТВОРЕННЯ ОБ'ЯВ З ПРОДАЖУ ТЗ ТА ЗАПЧАСТИН

3.1. Створення інтерфейсу додатку

При створенні інтерфейсу та UI елементів додатку розробник має керуватися такими принципами та правилами:

1) Простота і зрозумілість - інтерфейс повинен бути простим та легким для користувача, необхідно забезпечити зрозумілість елементів управління та взаємодії.

2) Навігація – створення легкості та зручності навігації між екранами та функціями.

3) Контекстно-залежний дизайн – дизайн та графічні елементи мають бути адаптивними до різних розмірів екранів та пристроїв в залежності від платформи IOS/Android.

4) Якість графічного дизайну – використовування зручних та естетично приємних кольорів та шрифтів. Забезпечення чіткості та читабельності тексту та інших елементів.

3.1.1. Відображення та кешування зображень

Оскільки додаток спеціалізується на розміщені об'яв, найважливішим графічним елементом у додатку є зображення ТЗ та деталей, тому процеси їх обробки є край важливими, а саме:

1) Завантаження.
2) Кешування.
3) Відображення.
4) Масштабування.

При завантаженні зображень при створенні запису ТЗ або запчастини, виставлене налаштування для завантаження фото із зменшеною якістю зображення на 30%, щоб запобігти розміщення у додатку зображень із великим розширенням та вагою, але такі міри не завжди є ефективними. На такий випадок створено структуру даних ImageStruct:

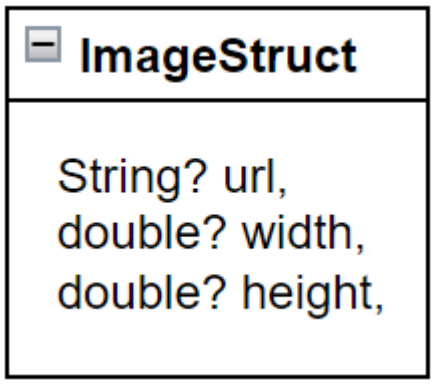

Рис.3.1. Зображення структури даних для зображень

Ця структура даних(рис.3.1) використовується під час завантаження даних зображення до бази даних, та при розміщенні зображень. URL – це посилання на зображення що розміщене у сховищі проекту додатку на Firebase, довжина та ширина необхідна для масштабування зображення відповідно до контейнерів у котрих знаходитиметься зображення як за розміром так і за вагою. Стандартним інструментом розміщення зображень з мережі у Flutter є віджет Image, а саме Image.network. Цей інструмент добре підходить для отримання одного чи декількох зображень із невеликим розміром, але у такому додатку що постійно оперує великою кількістю зображень, у котрих може бути різне розширення та вага він не підходить, та буде перевантажувати оперативну пам’ять. Гарним рішенням буде використовувати сторонній віджет CachedNetworkImage та супутній клас CachedImagesProvider. Ці інструменти предоставляють широкий спектр корисних функцій, але знадобиться лише декілька:

1) BoxFit – вказує як вписати зображення в простір, виділений під час верстки.

2) memCacheHeight та memCacheWidth - змінить розмір зображення в пам'яті, щоб воно мало певну висоту за допомогою [ResizeImage].

Ці параметри необхідні для того щоб роміщувати зображення із різним розширенням у контейнерах, так що б вони повністю їх заповнювали та підходили за розміром. Параметри memCacheHeight та memCacheWidth впливають на розмір зображення, що буде розміщене, ці параметри доцільно використовувати щоб усереднити розмір зображення до певного значення, котре буде балансувати між розміром пам’яті виділеним у пам’ять пристрою на зображення, та його якістю.

```
child: LayoutBuilder(
  builder: (context, constraints) {
    if (widget.toy!.images[1].height /
            widget.toy!.images[1].width >=
        1.2) {
      return getCachedNetworkImage(
          widget.toy!.images[1].url,
          (widget.toy!.images[1].height *
                  0.45)
              .toInt(),
          (widget.toy!.images[1].width *
                  0.45)
              .toInt(),
          BoxFit.fitWidth);
    } else if (widget.toy!.images[1].width /
            widget.toy!.images[1].height >=
        1.2) {
      return getCachedNetworkImage(
          widget.toy!.images[1].url,
          (widget.toy!.images[1].height *
                  0.45)
              .toInt(),
          (widget.toy!.images[1].width *
                  0.45)
              .toInt(),
          BoxFit.fitHeight);
    } else {
      return CachedNetworkImage(
        fadeInDuration:
            Duration(milliseconds: 0),
        fadeOutDuration:
            Duration(milliseconds: 0),
        imageUrl: widget.toy!.images[1].url,
        fit: BoxFit.fitWidth,
        memCacheHeight:
            (widget.toy!.images[1].height *
                    0.45)
                .toInt(),
        memCacheWidth:
            (widget.toy!.images[1].width *
                    0.45)
                .toInt(),
      ); // CachedNetworkImage
    }
  },
), // LayoutBuilder
```

Рис.3.2. Приклад коду що обробляє зображення

Віджет CachedNetworkImage дозволяє один раз завантажити зображення, та використовувати його у подальшому без повторного отримання його із мережі, що знизить навантаження на оперативну пам’ять та мережу пристрою.

На (Рис.3.2) зображений приклад коду що розміщує кешоване зображення у контейнері вираховуючи його пропорції відповідно до контейнеру та пропорцій самого зображення. Також зменшує його початковий розмір до 45% для запобігання перевищення використання оперативної пам’яті пристрою.

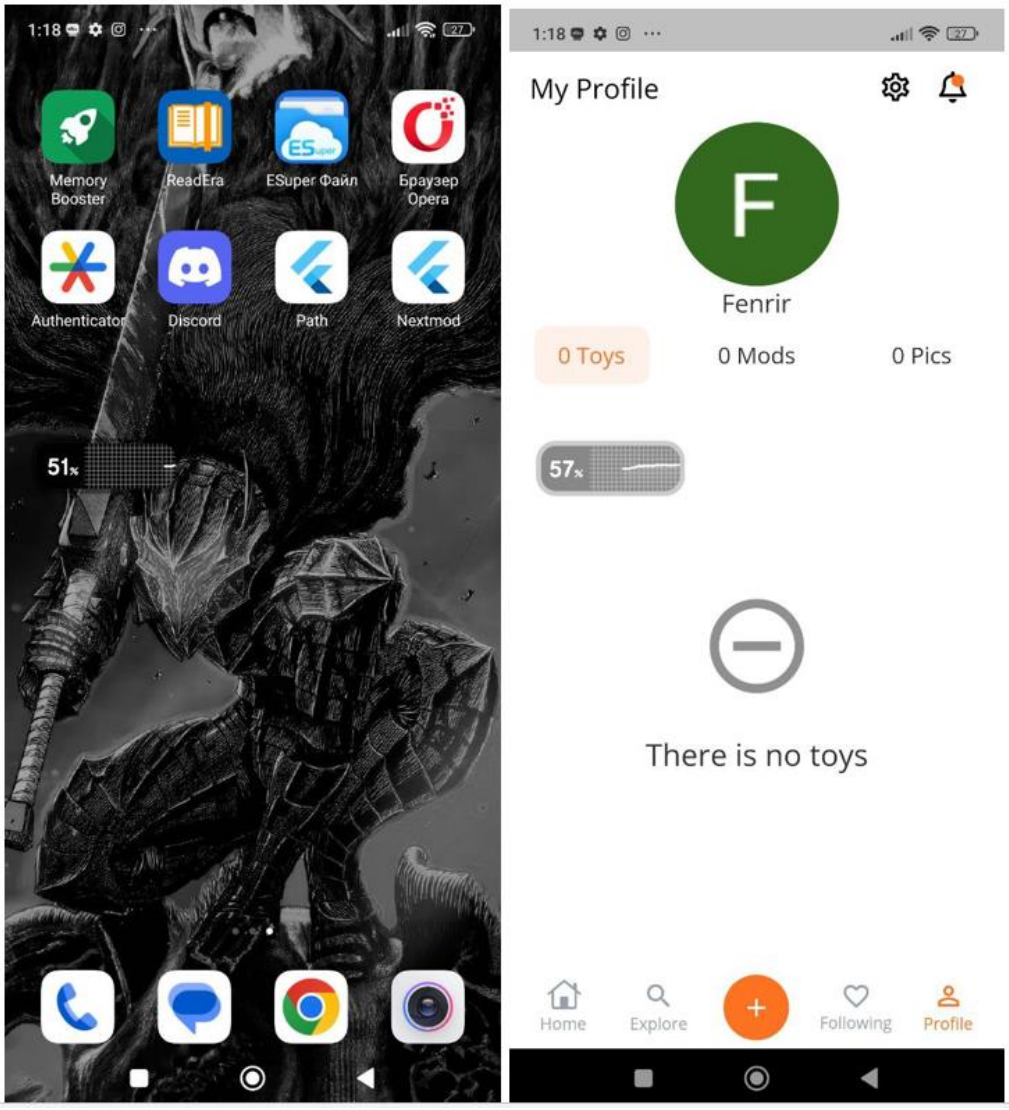

Рис.3.3. Результати тесту RAM

До впровадження використання CachedNewtorkImage додаток переповнював оперативну пам’ять за 2 хвилини активного використання, на даний момент він використовує 7% від загального об’єму(рис.3.3) (процент залежить від розміру оперативної пам’яті). Тестування проводилося на мобільному пристрої Xiaomi Redmi Note 12 Pro із розміром оперативної пам’яті 8Гб.

3.2. Створення логіки та графічних елементів для коментарів до ТЗ та деталей

Коментарі у такому типі додатків є необхідністю, оскільки вони допомагають користувачам прийняти обґрунтоване рішення про придбання. Коли користувач шукає транспортний засіб або запчастину, він хоче отримати якомога більше інформації про об'єкт продажу. Коментарі можуть надати користувачам додаткові відомості про стан об'єкта, його особливості та переваги. Наприклад, користувач може дізнатися від інших покупців про досвід використання транспортного засобу або запчастини, або може запитати у продавця про додаткову інформацію про об'єкт, який його цікавить.

Крім того, коментарі допомагають користувачам взаємодіяти з продавцями та іншими користувачами. Це може бути корисно для тих хто хоче задати запитання про об'єкт продажу, обговорити деталі покупки або просто встановити зв'язок з продавцем.

Також коментарі допомагають продавцям отримувати відгуки про свої ТЗ. Це може допомогти продавцям покращити свої об'яви та зробити їх більш привабливими для покупців. Об’єкт коментарю до ТЗ матиме вид(рис.3.4):

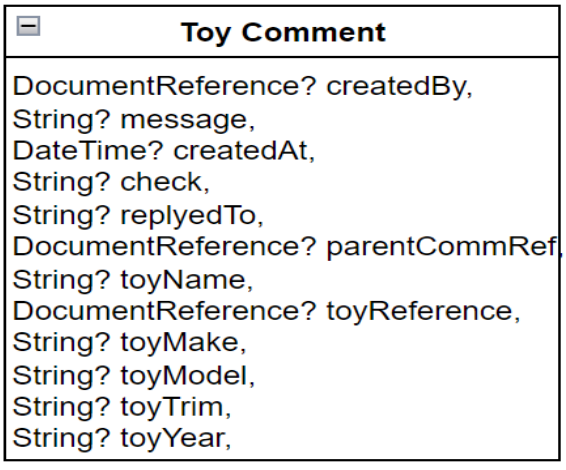

Рис.3.4. Структура даних коментарю до ТЗ

Він містить посилання на користувача що створив його, текст повідомлення, час створення, змінну флаг що використовується для відображення дочірнього та батьківського коментарю, посилання на батьківський комент, назву ТЗ та посилання на нього.

Такий вид матиме документ у БД, у підколекції toy-comments для ТЗ, а віджет у додатку:

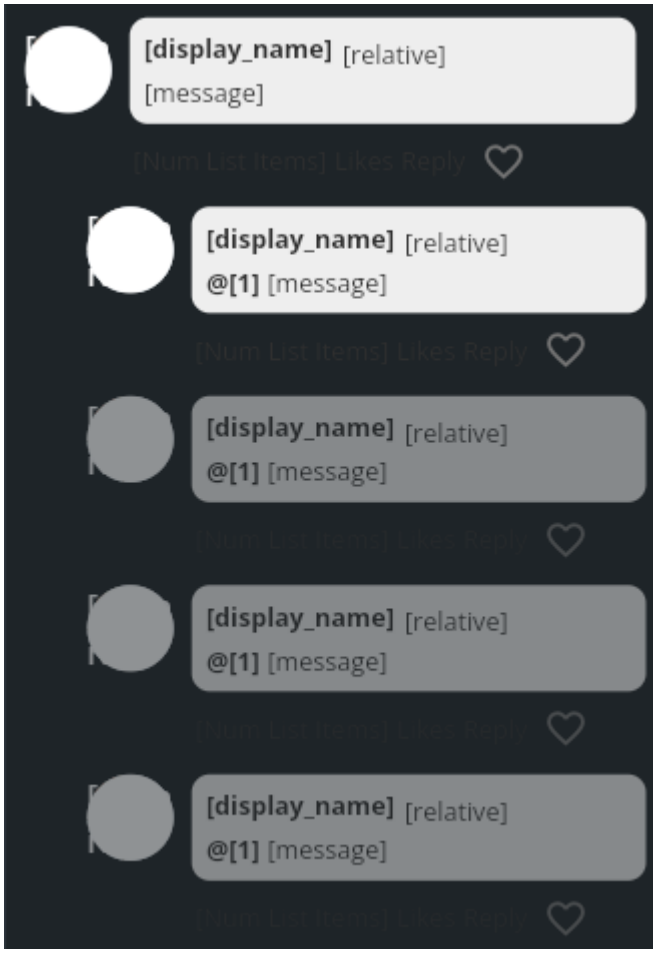

Рис.3.5. UI-елемент коментарю

На(рис.3.5) зображений каркас віджету, що буде заповнюватися даними при створенні коментарю. Найвищий коментар є батьківським, та має певне значення у змінній прапорі «check». Під батьківським коментарем знаходяться дочірні, що матимуть посилання на батьківський коментар, та відповідне значення у змінній прапорі. Цей віджет складається з двох трьох основних частин: контейнеру що вміщує у собі усі елементи, контейнеру що вміщує елемент батьківського коментарю(у самому верху) та колонку із дочірніми коментарями.

Для коментарів доступий перелік дій(рис.3.6), для покращення користувацького досвіду та збільшення функціональності додатку. Редагування коментарю, відповідь, копіювання та видалення:

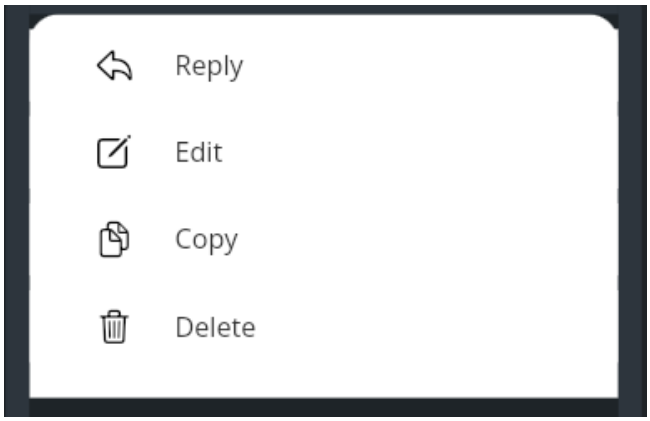

Рис.3.6. Список дій доступних для коментаря

1) Reply.

Для реалізації функціоналу відповіді на коментар використовується зміна стану сторінки(рис.3.7), за замовчуванням:

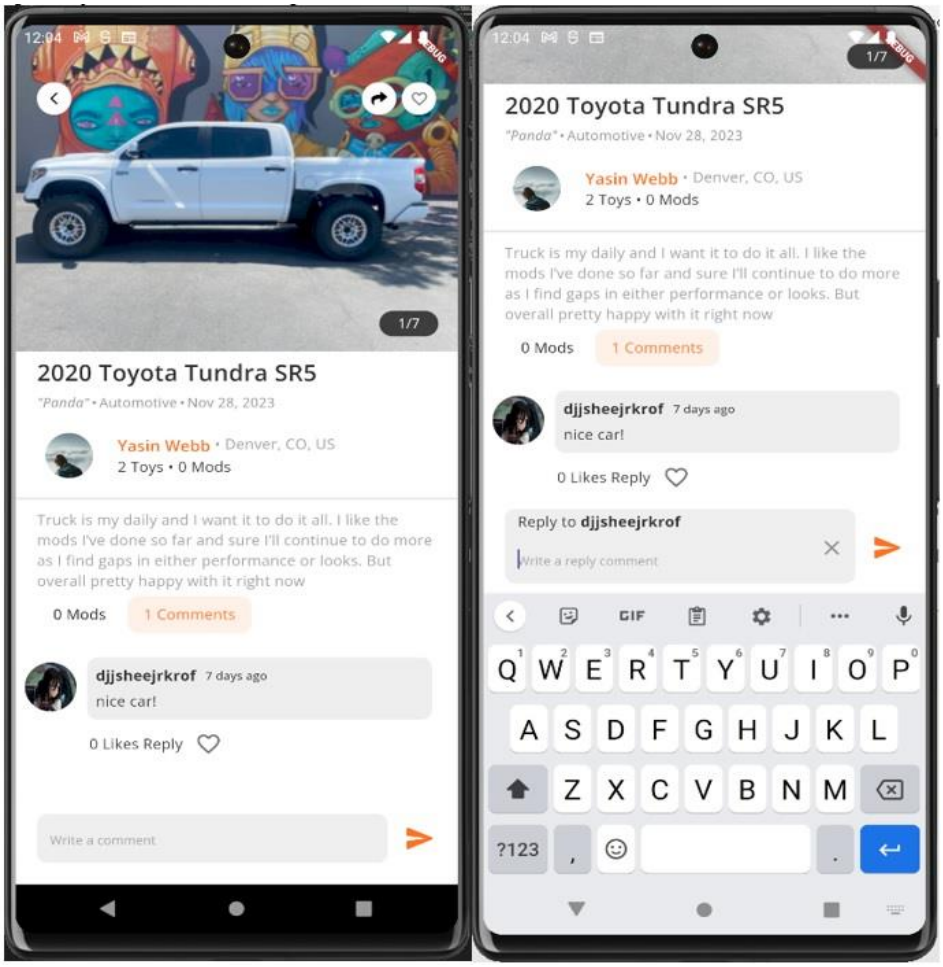

Рис.3.7. Зміна стану сторінки для відповіді на коментар

Стандартне поле вводу змінюється, додається відображення імені користувача, до котрого йде звернення та відображається іконка хрестик

відміни дії. Після вводу повідомлення необхідно натиснути на кнопку відправки, що створить дочірній документ коментарю у підколекції.

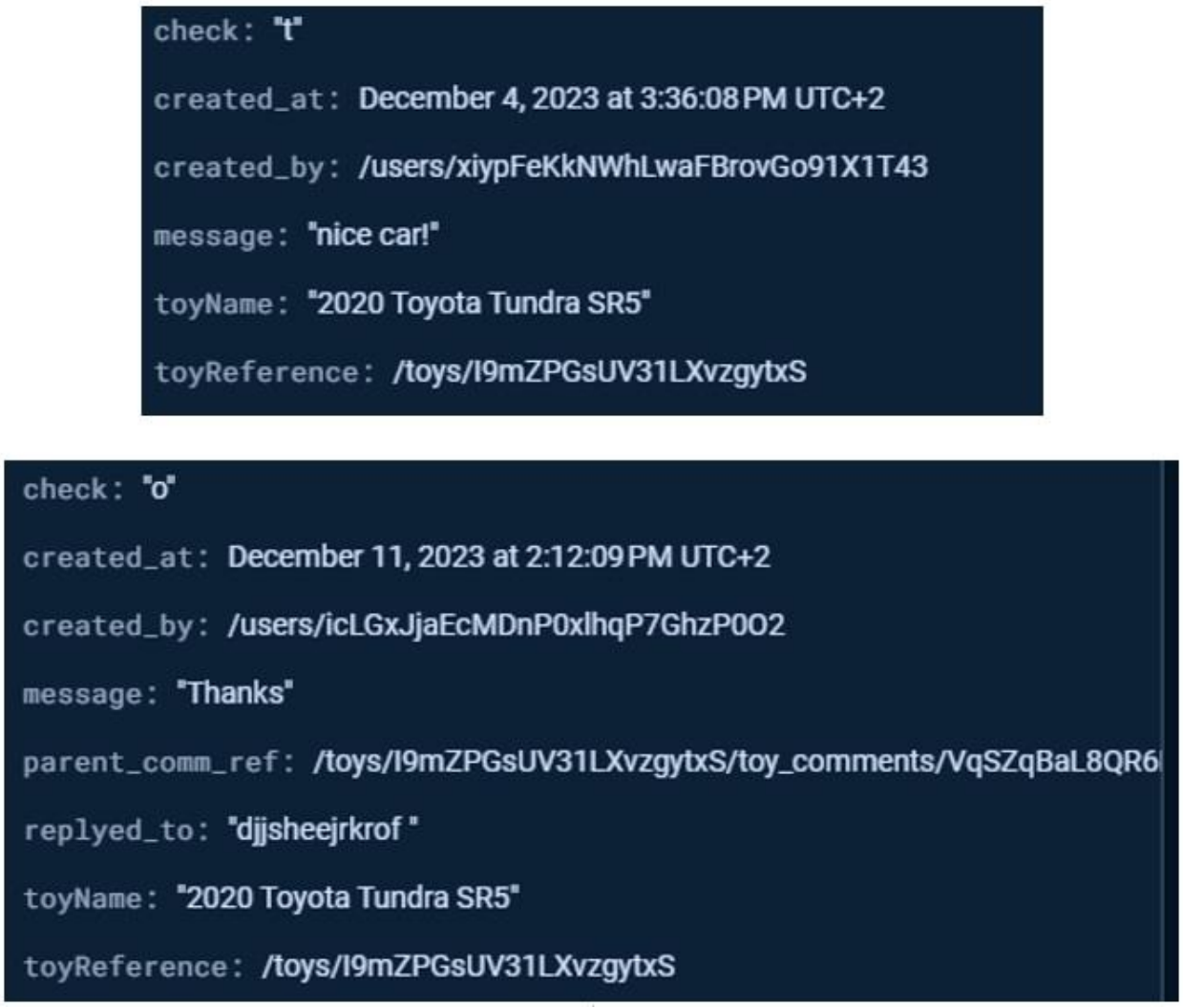

Рис.3.8. Порівняння дочірнього та батьківського коментарів

На(рис.3.8) зображена різниця між батьківським коментарем та його дочірнім. У дочірньому наявні декілька нових полів: посилання на батьківський коментар, ім’я користувача до котрого йде звернення та поле змінної прапору.

2) Редагування.

Коли користувач обирає функцію редагування свого коментарю також змінюється стан поточної сторінки, з’являється поле для вводу тексту що замінить тий що є у БД.

3) Копіювання.

При виборі функції копіювання, текст обраного коментарю буде скопійований у буфер обміну пристрою.

4) Видалення.

За допомогою цієї функції користувач може видалити свій створений коментар. Для випадків коли коментар є батьківським, будуть видалені й усі дочірні коментарі.

3.3. Створення Push-повідомлень у додатку, графічних елементів та логіки

Push-повідомлення складаються з наступних елементів [17]:

Заголовок - це короткий рядок тексту, який використовується для привернення уваги користувача.

Текст - це основний вміст push-повідомлення. Він може містити інформацію про нову функцію, розпродаж або іншу важливу подію.

Іконка додатка - це зображення, яке використовується для ідентифікації додатка, який надсилає push-повідомлення.

Дія - це кнопка або посилання, яке користувач може натиснути, щоб дізнатися більше про push-повідомлення або виконати певну дію.

Push-повідомлення можуть бути налаштовані таким чином, щоб вони відображалися на різних пристроях і в різних ситуаціях. Наприклад, push-повідомлення можуть бути налаштовані так, щоб вони відображалися тільки тоді, коли пристрій знаходиться в режимі "онлайн" або тільки тоді, коли користувач знаходиться в певній географічній зоні.

У додатку створено Push-нотифікації для випадків:

1) Відстеження та вподобання.

Оскільки у додатку існує можливість відстеження певних сутностей – ТЗ, користувачів, запчастин, з ними можуть відбуватися певні дії. Якщо один користувач відстежує іншого, він буде отримувати повідомлення під час його певних дій: створення запису із новим ТЗ, створення запису запчастини. Повідомлення будуть надходити також у випадках коли інший користувач

натиснув на іконку відстеження у вашому профілі, на сторінці вашого ТЗ або запчастини, вподобав ваш коментар.

2) Створення.

Також нотифікації створюватимуться коли користувач, котрого ви відстежуєте створює новий запис. Нотифікації спрацьовують при створенні коментарів до вашого запису.

Для створення функціоналу Push-повідомлень, використовується функціонал впроваджений FlutterFlow(рис.3.9):

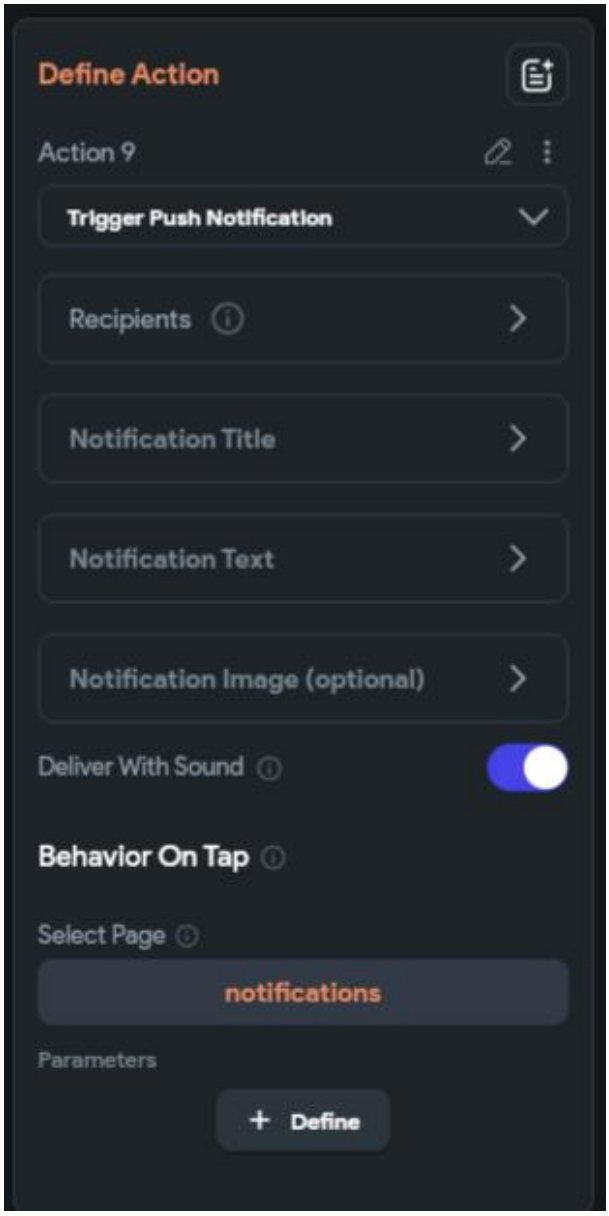

Рис.3.9. Параметри необхідні для створення повідомлення

Параметри що необхідно передати для створення повідомлення:

1) Одержувачі – у ролі одержувачів повідомлення виступатимуть люди що містяться у списку “followers” для певної сутності.

2) Заголовок – у заголовку буде відображений тип дії, що спричинила відправлення повідомлення: створення, коментар, вподобання.

3) Текст – текст міститиме опис дії що спричинила повідомлення.

4) Зображення – опціональний параметр, у ролі зображення у повідомленні може виступати аватар користувача що створив повідомлення.

1) Сторінка – при натисканні на віджет повідомлення, користувача перенесе до сторінки, де створено повідомлення.

Зокрема поодиноких повідомлень у додатку створена сторінка з усіма повідомленнями, що прийшли користувачу:

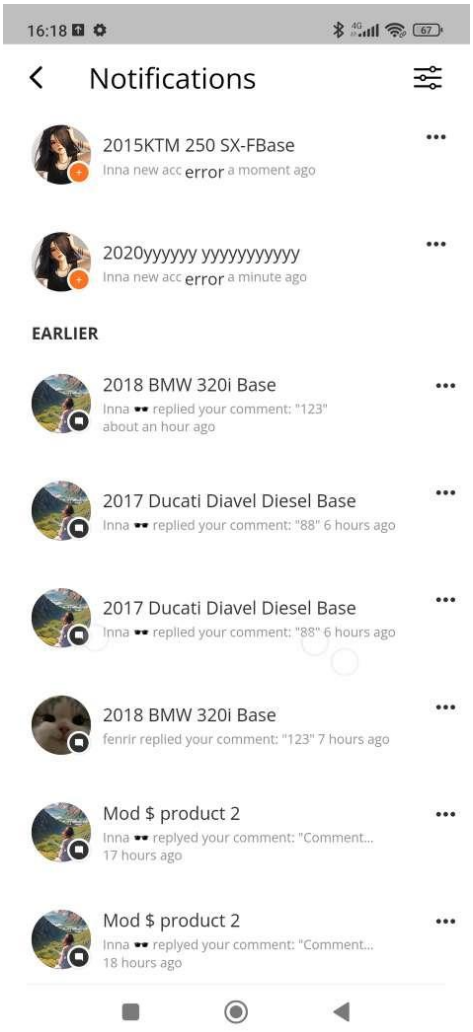

Рис.3.10. Сторінка із повідомленнями

Повідомлення розділені на прочитані та нові у два списки(рис.3.10), також для них доступні певні дії для покращення досвіду користувачів(рис.3.11):

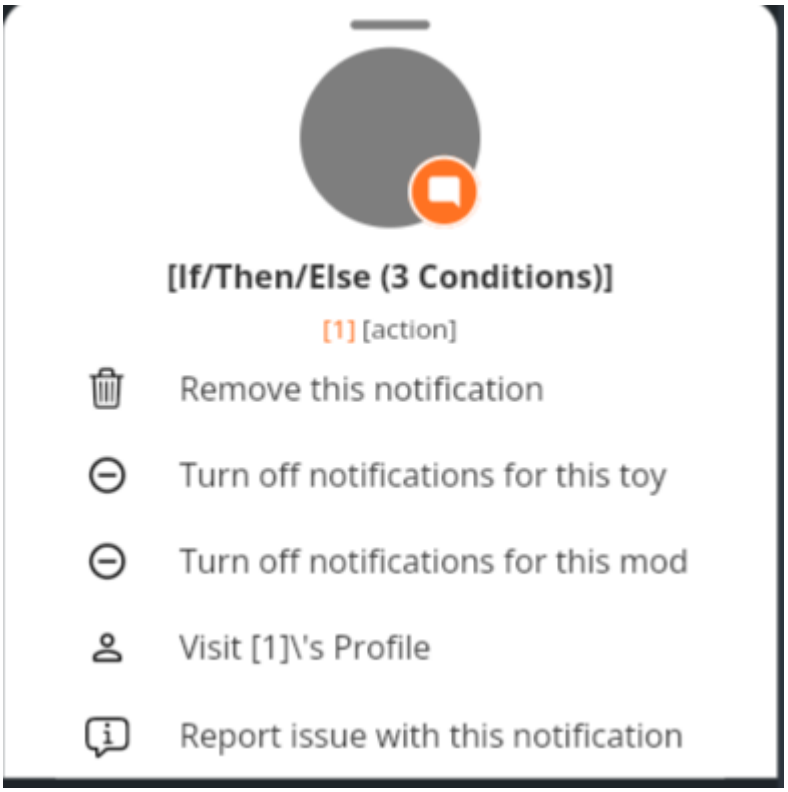

Рис.3.11. Дії доступні для повідомлення

1) Видалення повідомлення.
2) Вимкнення повідомлень для цього ТЗ або запчастини.
3) Відвідати профіль користувача від котрого прийшло повідомлення.
4) Поскаржитися на повідомлення.

3.4. Створення блокування користувачів та скарг на ТЗ, користувачів, запчастини.

У додатках із великою кількістю людей можуть знаходитися недоброчесні персони котрі спеціально будуть намагатися погіршити досвіт користування додатком для інших користувачів різними способами. Такими як лайка або нецензурні зображення, що призведе до негативної реакції певних людей. Для огородження потенціально недоброзичливих користувачів від

інших можна користуватися мануальною модерацією, але такі методи не досить ефективні через великий час обробки скарг адміністраторами додатку.

Тому доцільно буде створити блокування користувачів іншими користувачами та функціонал для скарг.

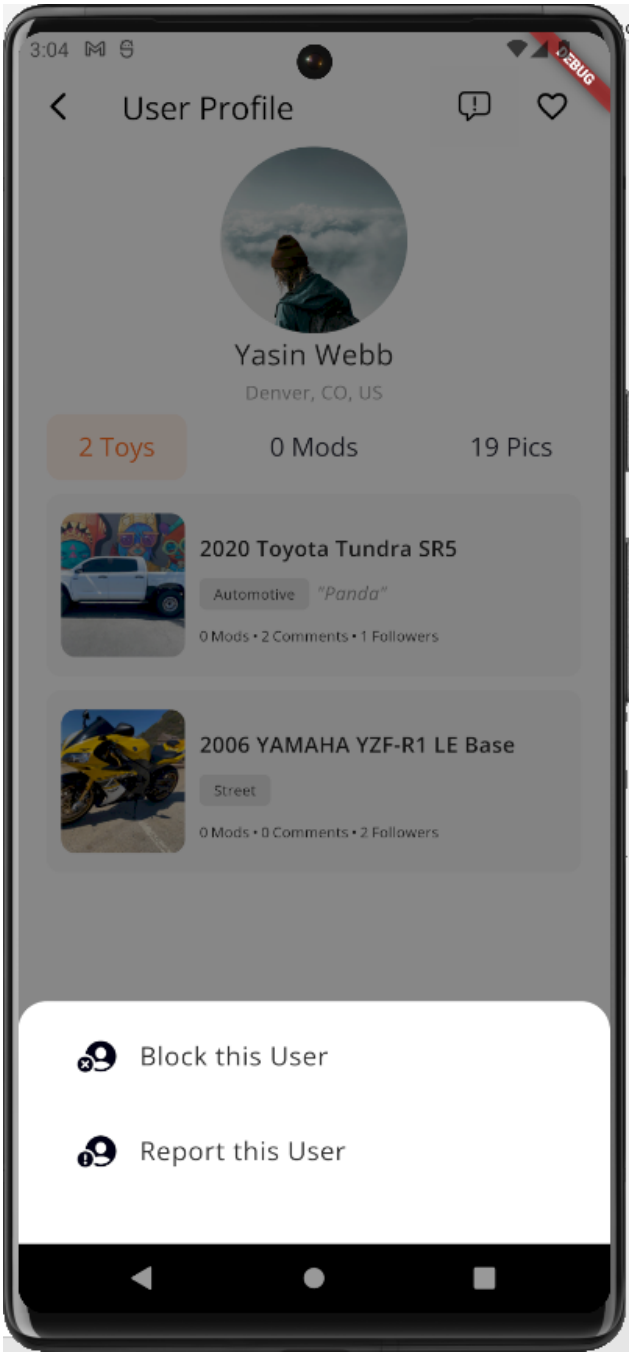

Рис.3.12. Список дій зі скаргою та блокуванням

На сторінці профілю користувача створено функціонал що надасть можливість іншим користувачам скаржитися на нього, та блокувати(рис.3.12).

При кількості скарг на користувача більше ніж 20, у його профілі з’явиться іконка попередження із текстом, що інформує про те що кристувач має велику кількість скарг.

Після натискання кнопки блокування, оновлюються документи обох користувачів у базі даних, а саме поля blockedUsers - для користувача що заблокував, та blockedBy – для заблокованого користувача.

У подальшому експлуатуванні додатку для обох користувачів буде взаємно виключений з усіх списків контент що відноситься до заблокованого користувача, та того хто заблокував.

```
List<ToyCommentsRecord>
    textToyCommentsRecordList =
    snapshot.data!;
textToyCommentsRecordList.removeWhere(
    (element) => currentUserDocument!
        .blockedBy
        .contains(element.createdBy));
textToyCommentsRecordList.removeWhere(
    (element) => currentUserDocument!
        .blockedUsers
        .contains(element.createdBy));
```

Рис.3.13. Приклад коду що редагує список

Процес виключення даних(рис.3.13) із усіх списків відбуватиметься під час їх генерації. Оскільки ми не можемо зробити фільтрацію даних одразу при отриманні документів з колекції, оскільки Firebase не дозволяє більш ніж один фільтр WHERE-IN, то виключимо самостійно дані. Для цього звернемося до списку отриманих та перетворених до локальних структур даних записів, а після використовуючи метод removeWhere виключимо усі дані зі списку, котрі не створені заблокованими користувачами, також виключаються дані тих користувачів що заблокували вас. Після усіх цих процесів буде згенерований список із усіма відкоректованими даними.

Також доцільно буде створити сторінку(рис.3.14) із де будуть відображатися усі заблоковані вами користувачі для подальшого їх розблокування.

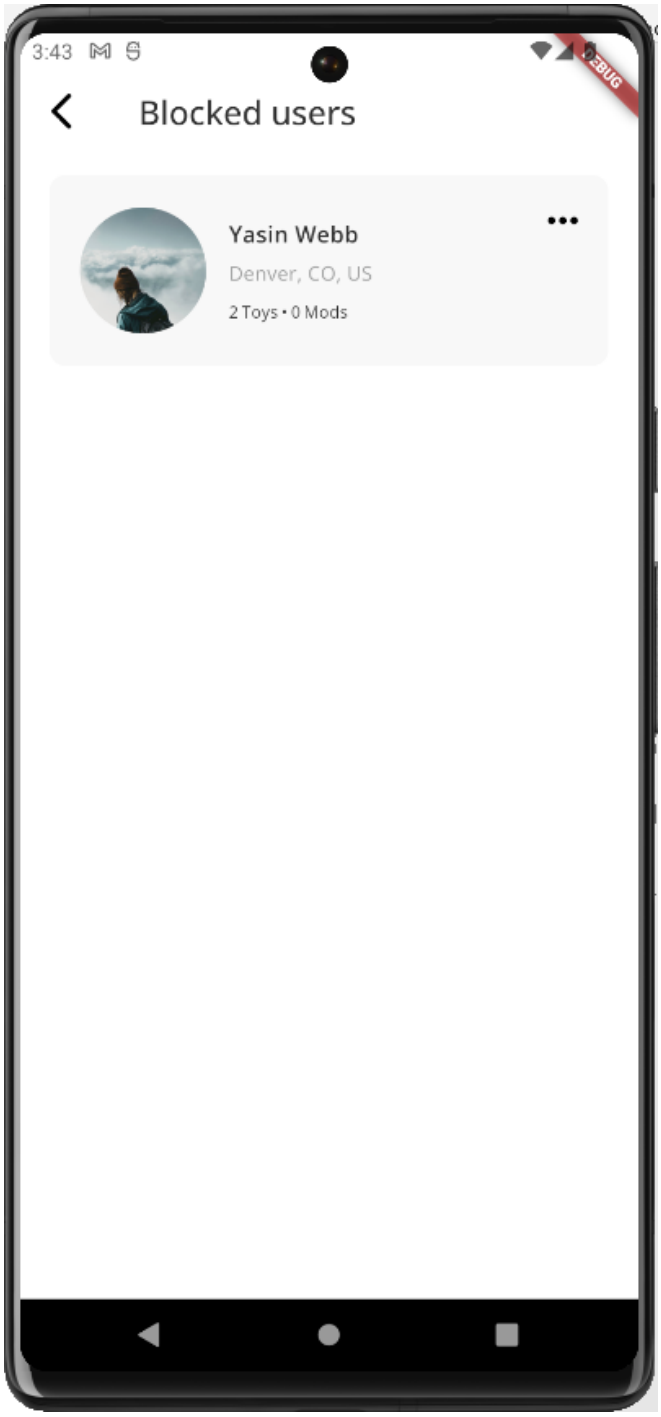

Рис.3.14. Сторінка із заблокованими користувачами

ВИСНОВОК

В ході даної кваліфікаційної роботи магістра був створений мобільний кросплатформний додаток для розміщення об'яв з продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter. Були розглянуті аспекти ринків ТЗ та мобільних додатків у світі та Україні, порівняні веб-додатки та мобільні додатки. Також була розроблена структура нереляційної бази даних для додатку. Був створений новаторський алгоритм фільтрації даних що забезпечує мінімальне використання ресурсів мобільного пристрою та мінімізує навантаження на інтернет-трафік. Розроблений інтерфейс додатку що не перевантажений графічними елементами та зручний у використанні.

Наукова новизна отриманих результатів дипломної роботи визначається тим, що вперше розроблено і обґрунтовано інформаційну технологію побудови мобільного кросплатформного додатку для розміщення об'яв з продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter, шляхом спорщення процесу розробки завдяки використанню двох сервісів для розробки та створенням нових алгоритмів для фільтрації та розміщення даних всередині інформаційної системи, що покращує продуктивність додатку.

Практична цінність результатів полягає у тому, що розроблена інформаційна система кросплатформного додатку надає можливість використання платформи для створення об’яв з продажу ТЗ та запчастин до них.

Таким чином мета кваліфікаційної роботи досягнута в повному обсязі, крім того додаток завантажено до платформи Google Play [18].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільний додаток [Електронний ресурс]/URL : https://ru.wikipedia.org/wiki/ Мобільний додаток.
2. The History of Mobile Apps [Електронний ресурс]/URL : https://inventionland.com/blog/the-history-of-mobile-apps/
3. Загальний графік світових продаж ТЗ [Електронний ресурс]/URL : https://fred.stlouisfed.org/.
4. Commercial Vehicles Market [Електронний ресурс]/URL : https://www.precedenceresearch.com/commercial-vehicles-market.
5. Ринок вживаних авто дослідження [Електронний ресурс]/URL : https://eauto.org.ua/static/documents/Ринок_вживаних_авто_дослідження
6. ТОП-15 МАРКЕТПЛЕЙСІВ УКРАЇНИ [Електронний ресурс]/URL : https://uba.top/marketplaces/.
7. Мобільні додатки будуть визначати майбутнє бізнесу [Електронний ресурс]/URL : https://techcrunch.com/2010/04/12/eric-schmidt-mobile-is-the-future-and-theres-no-such-thing-as-communication-overload/.
8. Mobile app downloads worldwide from 1st half of 2019 to 1st half of 2023 [Електронний ресурс]/URL : https://www.statista.com/statistics/1401281/app-global-downloads/.
9. Number of mobile app downloads worldwide from 2019 to 2027, by segment [Електронний ресурс]/URL : https://www.statista.com/forecasts/1262892/mobile-app-revenue-worldwide-by-segment/.
10. Flutter architectural overview [Електронний ресурс]/URL : https://docs.flutter.dev/resources/architectural-overview.
11. Cross Platform [Електронний ресурс]/URL : https://www.techopedia.com/definition/17056/cross-platform.
12. Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022 [Електронний ресурс]/URL :

https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.

13. FlutterFlow: A Quick Overview [Електронний ресурс]/URL : https://medium.com/@zipper_triumph/flutterflow-a-comprehensive-guide-45f780375e4e/.
14. Firebase – Introduction [Електронний ресурс]/URL : https://www.geeksforgeeks.org/firebase-introduction/.
15. Difference Between MVC, MVP and MVVM Architecture Pattern in Android [Електронний ресурс]/URL : https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/.
16. Flutter Firebase Auth Login [Електронний ресурс]/URL : https://www.youtube.com/watch?app=desktop&v=hgRg9RFvNJQ.
17. Push Notifications: The Ultimate Guide [Електронний ресурс]/URL : https://onesignal.com/what-are-push-notifications.
18. NextModApp [Електронний ресурс]/URL : (https://play.google.com/store/apps/details?id=nextmodapp.com&hl=ru&gl=US).

ДОДАТОК А

Програмний код (лістинг) компонентів застосунку

home_page_widget.dart

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:nextmod/widget_helpers/skeleton.dart';

import '/auth/firebase_auth/auth_util.dart';
import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_theme.dart';
import '/flutter_flow/flutter_flow_util.dart';
import
'/widget_helpers/custom_nav_bar/custom_nav_bar_widget.dart';
import
'/widget_helpers/history/history_mod_component/history_mod_compon
ent_widget.dart';
import
'/widget_helpers/history/history_toy_component/history_toy_component
_widget.dart';
import '/custom_code/actions/index.dart' as actions;
import 'package:flutter/material.dart';
import 'package:flutter/scheduler.dart';
import 'package:flutter_spinkit/flutter_spinkit.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:provider/provider.dart';

import 'home_page_model.dart';

class HomePageWidget extends StatefulWidget {
  const HomePageWidget({Key? key}) : super(key: key);

  @override
  _HomePageWidgetState createState() =>
_HomePageWidgetState();
}
```

```
class _HomePageWidgetState extends State<HomePageWidget> {
  late HomePageModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();


  @override
  void initState() {
    super.initState();
    _model = createModel(context, () => HomePageModel());

    logFirebaseEvent('screen_view', parameters: {'screen_name':
'home_page'});
    // On page load action.
    SchedulerBinding.instance.addPostFrameCallback((_) async {

logFirebaseEvent('HOME_PAGE_PAGE_home_page_ON_INIT_STAT
E');
      if (currentUserDocument!.displayName == '') {
        _model.photoLibraryAccess = await
actions.isPhotoLibraryAccess();
        logFirebaseEvent('home_page_update_app_state');
        setState(() {
          FFAppState().notificationsAcces =
_model.notificationGranted!;
          FFAppState().photoLibraryAccess =
_model.photoLibraryAccess!;
        });
      } else {
        logFirebaseEvent('home_page_navigate_to');

        context.pushNamed(
          'verify_emailUnset',
          queryParameters: {
            'userEmail': serializeParam(
              currentUserEmail,
              ParamType.String,
            ),
          }.withoutNulls,
```

```
        );
      }
    });


  }



  void scrollListener() async {
    }
  }
  }

  Future<void> getToys() async {
    try {

      isLoading = true;

    var data = await FirebaseFirestore.instance
        .collection('history_new')
        .where('user_ref', isNotEqualTo: currentUserReference)
        .orderBy('user_ref')
        .limit(5)
        .get();

    if (data.docs.isNotEmpty) {
      pagination = data.docs.last;
    }

      testToys = data.docs
          .map((e) =>
              HistoryNewRecord.getDocumentFromData(e.data(),
e.reference))
          .where((element) =>

currentUserDocument!.blockedUsers.contains(element.userRef) ==
                  false &&
```

```
currentUserDocument!.blockedBy.contains(element.userRef) ==
                    false)
                .toList();

            if (testToys.isEmpty || testToys.length < 5) {
              getNextToys();
            }

          } finally {

            isLoading = false;

          }
        }

        Future<void> getNextToys() async {
          try {

            isLoading = true;

            var data = await FirebaseFirestore.instance
                .collection('history_new')
                .where('user_ref', isNotEqualTo: currentUserReference)
                .orderBy('user_ref')
                .limit(5)
                .startAfterDocument(pagination!)
                .get();

            if (data.docs.isNotEmpty) {
              pagination = data.docs.last;


              testToys.addAll(data.docs
                  .map((e) =>
                      HistoryNewRecord.getDocumentFromData(e.data(),
e.reference))
                  .where((element) =>
```

```
currentUserDocument!.blockedUsers.contains(element.userRef) ==
                    false &&

currentUserDocument!.blockedBy.contains(element.userRef) ==
                    false)
                .toList());


            if (testToys.isEmpty) {
              getNextToys();
            }

          }

        } finally {

            isLoading = false;

        }
      }

      @override
      void dispose() {

        _model.dispose();
        scrollController.removeListener(scrollListener);

        super.dispose();
      }

      @override
      Widget build(BuildContext context) {
        context.watch<FFAppState>();
        Size size = MediaQuery.of(context).size;
        return GestureDetector(
          onTap: () =>
FocusScope.of(context).requestFocus(_model.unfocusNode),
          child: Scaffold(
```

```
            key: scaffoldKey,
            backgroundColor: Colors.white,
            appBar: responsiveVisibility(
              context: context,

            )
                ? AppBar(
                  scrolledUnderElevation: 0,
                    backgroundColor: Colors.white,
                    automaticallyImplyLeading: false,
                    title: Text(
                      'Home',
                      style:
FlutterFlowTheme.of(context).headlineMedium.override(
                            fontFamily:

FlutterFlowTheme.of(context).headlineMediumFamily,
                            color: Colors.black,
                            fontSize: 22,
                            useGoogleFonts: GoogleFonts.asMap().containsKey(

FlutterFlowTheme.of(context).headlineMediumFamily),
                          ),
                    ),
                    actions: [
                      Row(
                        mainAxisSize: MainAxisSize.max,
                        children: [
                          Padding(
                            padding: EdgeInsetsDirectional.fromSTEB(0, 0, 5,
0),
                            child: InkWell(
                              splashColor: Colors.transparent,
                              focusColor: Colors.transparent,
                              hoverColor: Colors.transparent,
                              highlightColor: Colors.transparent,
                              onTap: () async {
                                logFirebaseEvent(
```

```
'HOME_PAGE_PAGE_Container_n5ao016s_ON_TAP');
                        logFirebaseEvent('Container_navigate_to');

                        context.pushNamed(
                          'explore_page',
                          queryParameters: {
                            'isSearch': serializeParam(
                              true,
                              ParamType.bool,
                            ),
                          }.withoutNulls,
                        );
                      },
                      child: Container(
                        width: 50,
                        height: 30,
                        decoration: BoxDecoration(
                          color: FlutterFlowTheme.of(context)
                              .secondaryBackground,
                        ),
                        child: Icon(
                          Icons.search_rounded,
                          color: Colors.black,
                          size: 30,
                        ),
                      ),
                    ),
                  ),
                  FutureBuilder<int>(
                    future: queryNotificationsRecordCount(
                      queryBuilder: (notificationsRecord) =>
                          notificationsRecord
                              .where('recievers',
                                  arrayContains: currentUserReference)
                              .where('seen', isEqualTo: false),
                    ),
                    builder: (context, snapshot) {
```

```
                    // Customize what your widget looks like when it's
loading.
                    if (!snapshot.hasData) {
                      return Center(
                        child: Container(
                          width: 50,
                          height: 50,
                          decoration: BoxDecoration(
                            color: Color(0x00FFFFFF),
                          ),
                          child: Align(
                            alignment: AlignmentDirectional(0, 0),
                            child: FaIcon(
                              FontAwesomeIcons.bell,
                              color: Colors.black,
                              size: 25,
                            ),
                          ),
                        ),
                      );
                    }
                    int stackCount = snapshot.data!;
                    return Stack(
                      alignment: AlignmentDirectional(
                          0.19999999999999996, -0.25),
                      children: [
                        InkWell(
                          splashColor: Colors.transparent,
                          focusColor: Colors.transparent,
                          hoverColor: Colors.transparent,
                          highlightColor: Colors.transparent,
                          onTap: () async {
                            logFirebaseEvent(

'HOME_PAGE_PAGE_Container_vjtq23dl_ON_TAP');
                            logFirebaseEvent('Container_navigate_to');

                            context.pushNamed('notifications');
                          },
```

```
                child: Container(
                  width: 50,
                  height: 50,
                  decoration: BoxDecoration(
                    color: Color(0x00FFFFFF),
                  ),
                  child: Align(
                    alignment: AlignmentDirectional(0, 0),
                    child: FaIcon(
                      FontAwesomeIcons.bell,
                      color: Colors.black,
                      size: 25,
                    ),
                  ),
                ),
              ),
              if (valueOrDefault<bool>(
                stackCount > 0,
                true,
              ))
                Align(
                  alignment: AlignmentDirectional(1, -0.3),
                  child: Container(
                    width: 10,
                    height: 10,
                    decoration: BoxDecoration(
                      color: FlutterFlowTheme.of(context)
                          .mainOrange,
                      shape: BoxShape.circle,
                    ),
                  ),
                ),
            ],
          );
        },
      ),
    ],
  ),
],
```

```
                centerTitle: false,
                elevation: 0,
              )
            : null,
          body: Stack(
            children: [
              Padding(
                padding: EdgeInsetsDirectional.fromSTEB(0, 0, 0, 65),
                child: StreamBuilder<List<HistoryNewRecord>>(
                  stream: queryHistoryNewRecord(
                    queryBuilder: (historyNewRecord) =>
historyNewRecord
                        .where('user_ref', isNotEqualTo:
currentUserReference),
                  ),
                  builder: (context, snapshot) {
                    // Customize what your widget looks like when it's
loading.
                    if (!snapshot.hasData) {
                      return Center(
                        child: SizedBox(
                          width: 50.0,
                          height: 50.0,
                          child: SpinKitFadingCircle(
                            color: FlutterFlowTheme.of(context).mainOrange,
                            size: 50.0,
                          ),
                        ),
                      );
                    }

                      List<HistoryNewRecord>? history = snapshot.data;

                    return ListView.separated(
                      cacheExtent: size.height,

                      padding: EdgeInsets.zero,
                      scrollDirection: Axis.vertical,
                      separatorBuilder: (_, __) => SizedBox(height: 20),
```

```
                    itemCount: history!.length,
                    itemBuilder: (context, listViewIndex) {
                      final listViewHistoryNewRecord =
history[listViewIndex];
                      return Builder(
                        builder: (context) {
                          if (listViewHistoryNewRecord.hasToyRef()) {
                            return SizedBox(
                              width: size.width,
                              height: size.height * 0.36,
                              child: StreamBuilder<ToysRecord>(
                                stream: ToysRecord.getDocument(
                                    listViewHistoryNewRecord.toyRef!),
                                builder: (context, snapshot) {
                                  // Customize what your widget looks like when
it's loading.
                                  if (!snapshot.hasData) {
                                    return Center(
                                      child: Padding(
                                        padding: const EdgeInsets.fromLTRB(
                                            20, 15, 20, 0),
                                        child: HomeCardSkelton(),
                                      ),
                                    );
                                  }
                                  final historyToyComponentToysRecord =
                                      snapshot.data!;
                                  return InkWell(
                                    splashColor: Colors.transparent,
                                    focusColor: Colors.transparent,
                                    hoverColor: Colors.transparent,
                                    highlightColor: Colors.transparent,
                                    onTap: () async {
                                      logFirebaseEvent(

'HOME_PAGE_PAGE_Container_jzrxb1zd_ON_TAP');
                                      if (historyToyComponentToysRecord
                                              .createdBy ==
                                          currentUserReference) {
```

```
                                    logFirebaseEvent(
                                        'history_toy_component_navigate_to');
                                    context.pushNamed(
                                      'toy_page',
                                      queryParameters: {
                                        'toy': serializeParam(
                                          historyToyComponentToysRecord,
                                          ParamType.Document,
                                        ),
                                      }.withoutNulls,
                                      extra: <String, dynamic>{
                                        'toy':
                                            historyToyComponentToysRecord,
                                      },
                                    );
                                  } else {
                                    logFirebaseEvent(
                                        'history_toy_component_navigate_to');

                                    context.pushNamed(
                                      'user_toy_page',
                                      queryParameters: {
                                        'toy': serializeParam(
                                          historyToyComponentToysRecord,
                                          ParamType.Document,
                                        ),
                                      }.withoutNulls,
                                      extra: <String, dynamic>{
                                        'toy':
                                            historyToyComponentToysRecord,
                                      },
                                    );
                                  }
                                },
                                child: HistoryToyComponentWidget(
                                  key: Key(

'Keyjzr_${listViewIndex}_of_${testToys.length}'),
                                  userPhoto:
```

```
                        listViewHistoryNewRecord.userPhoto,
                    userLocation:
                        listViewHistoryNewRecord.userLocation,
                    userName:
                        listViewHistoryNewRecord.userName,
                    toy: historyToyComponentToysRecord,
                  ),
                );
              },
            ),
          );
        } else {
          return SizedBox(
            width: size.width,
            height: size.height * 0.36,
            child: StreamBuilder<ModsRecord>(
              stream: ModsRecord.getDocument(
                  listViewHistoryNewRecord.modRef!),
              builder: (context, snapshot) {
                // Customize what your widget looks like when
it's loading.
                if (!snapshot.hasData) {
                  return Center(
                    child: Padding(
                      padding: const EdgeInsets.fromLTRB(
                          20, 15, 20, 0),
                      child: HomeCardSkelton(),
                    ),
                  );
                }
                final historyModComponentModsRecord =
                    snapshot.data!;
                return InkWell(
                  splashColor: Colors.transparent,
                  focusColor: Colors.transparent,
                  hoverColor: Colors.transparent,
                  highlightColor: Colors.transparent,
                  onTap: () async {
                    logFirebaseEvent(
```

```
'HOME_PAGE_PAGE_Container_ykv7q6n4_ON_TAP');
                                    if (historyModComponentModsRecord
                                          .createdBy ==
                                      currentUserReference) {
                                     logFirebaseEvent(
                                        'history_mod_component_navigate_to');

                                     context.pushNamed(
                                      'mod_page',
                                      queryParameters: {
                                       'modDocument': serializeParam(
                                         historyModComponentModsRecord,
                                         ParamType.Document,
                                       ),
                                      }.withoutNulls,
                                      extra: <String, dynamic>{
                                       'modDocument':
                                           historyModComponentModsRecord,
                                      },
                                     );
                                    } else {
                                     logFirebaseEvent(
                                        'history_mod_component_navigate_to');

                                     context.pushNamed(
                                      'user_mod_page',
                                      queryParameters: {
                                       'mod': serializeParam(
                                         historyModComponentModsRecord,
                                         ParamType.Document,
                                       ),
                                      }.withoutNulls,
                                      extra: <String, dynamic>{
                                       'mod':
                                           historyModComponentModsRecord,
                                      },
                                     );
                                    }
```

```
                                    },
                                    child: HistoryModComponentWidget(
                                      key: Key(

'Keyykv_${listViewIndex}_of_${testToys.length}'),
                                      userPhoto:
                                          listViewHistoryNewRecord.userPhoto,
                                      userLocation:
                                          listViewHistoryNewRecord.userLocation,
                                      userName:
                                          listViewHistoryNewRecord.userName,
                                      mod: historyModComponentModsRecord,
                                    ),
                                  );
                                },
                              ),
                            );
                          }
                        },
                      );
                    },
                  );
                },
              ),
            ),
            Align(
              alignment: AlignmentDirectional(0, 1),
              child: wrapWithModel(
                model: _model.customNavBarModel,
                updateCallback: () => setState(() {}),
                child: CustomNavBarWidget(
                  homeColor: FlutterFlowTheme.of(context).mainOrange,
                  exploreColor:

              ),
            ),
          ],
        ),
      ),
```

```
    );
  }
}

//  return Center(
//                              child: SizedBox(
//                                width: 50,
//                                height: 50,
//                                child: SpinKitFadingCircle(
//                                  color: FlutterFlowTheme.of(context)
//                                      .mainOrange,
//                                  size: 50,
//                                ),
//                              ),
//                            );
```

settings_widget.dart

```
import 'dart:io';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/scheduler.dart';
import 'package:flutter/services.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:nextmod/custom_code/actions/is_photo_library_access.dart';
import
'package:nextmod/widget_helpers/profile/delete_user_dialog/delete_user_dial
og_widget.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:provider/provider.dart';
import 'package:share_plus/share_plus.dart';
```

```
import '/auth/firebase_auth/auth_util.dart';
import '/backend/backend.dart';
import '/flutter_flow/flutter_flow_theme.dart';
import '/flutter_flow/flutter_flow_util.dart';
import '/widget_helpers/settings/logout/logout_widget.dart';
import 'settings_model.dart';

export 'settings_model.dart';

class SettingsWidget extends StatefulWidget {
  const SettingsWidget({Key? key}) : super(key: key);

  @override
  _SettingsWidgetState createState() => _SettingsWidgetState();
}

class _SettingsWidgetState extends State<SettingsWidget> {
  late SettingsModel _model;
bool onPressed = true;
  final scaffoldKey = GlobalKey<ScaffoldState>();
  late PermissionStatus status;
  @override
  void initState() {
    super.initState();

    _model = createModel(context, () => SettingsModel());
    SchedulerBinding.instance.addPostFrameCallback((_) async {
      print(await Permission.storage.status);
      if (await Permission.storage.status.isDenied ||
          await Permission.storage.status.isPermanentlyDenied) {
```

```
      setState(() {
        FFAppState().photoLibraryAccess = false;
      });
    } else if (await Permission.storage.status.isGranted) {
      setState(() {
        FFAppState().photoLibraryAccess = true;
      });
    }

    setState(() async {
      _model.switchValue2 = await isPhotoLibraryAccess()
          ? null
          : FFAppState().photoLibraryAccess;
    });
  });

  logFirebaseEvent('screen_view', parameters: {'screen_name': 'settings'});
}

void openSettings() {
  // Use the permission_handler package to open app settings
  openAppSettings();
}

removePermissionDialog(context) => showCupertinoDialog<void>(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) => CupertinoAlertDialog(
        title: const Text('Permission Settings'),
        content: const Text('Disallow access to gallery and photos'),
```

```
      actions: <CupertinoDialogAction>[
        CupertinoDialogAction(
          onPressed: () => Navigator.of(context).pop(),
          child: const Text('Cancel'),
        ),
        CupertinoDialogAction(
          isDefaultAction: true,
          onPressed: () => openSettings(),
          child: const Text('Settings'),
        ),
      ],
    ),
  );

setPermissionDialog(context) => showCupertinoDialog<void>(
    context: context,
    barrierDismissible: false,
    builder: (BuildContext context) => CupertinoAlertDialog(
      title: const Text('Permission Settings'),
      content: const Text('Allow access to gallery and photos'),
      actions: <CupertinoDialogAction>[
        CupertinoDialogAction(
          onPressed: () => Navigator.of(context).pop(),
          child: const Text('Cancel'),
        ),
        CupertinoDialogAction(
          isDefaultAction: true,
          onPressed: () => openSettings(),
          child: const Text('Settings'),
        ),
```

```
        ],
      ),
    );

void showPermissionAlertDialog(BuildContext context) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text("Permission Required"),
        content:
            Text("Do you want to disallow the app from using your gallery?"),
        actions: <Widget>[
          // "Cancel" button
          TextButton(
            child: Text("Cancel"),
            onPressed: () {
              context.pop(); // Close the dialog
            },
          ),

          // "Open Settings" button
          TextButton(
            child: Text("Open Settings"),
            onPressed: () {
              try {
                openSettings(); // Open the app settings
                context.pop(); // Close the dialog
              } catch (e) {
                print(e);
```

```
            }
          },
        ),
      ],
    );
  },
 );
}

@override
void dispose() {
 _model.dispose();

 super.dispose();
}

@override
Widget build(BuildContext context) {
 context.watch<FFAppState>();

 return Scaffold(
   key: scaffoldKey,
   backgroundColor: Colors.white,
   appBar: responsiveVisibility(
     context: context,
   )
       ? AppBar(
           systemOverlayStyle: SystemUiOverlayStyle.dark,
           backgroundColor: Color(0x00FFFFFF),
           automaticallyImplyLeading: false,
```

```
          leading: Container(
            width: 100.0,
            height: 100.0,
            decoration: BoxDecoration(
              color: FlutterFlowTheme.of(context).secondaryBackground,
            ),
            child: InkWell(
              splashColor: Colors.transparent,
              focusColor: Colors.transparent,
              hoverColor: Colors.transparent,
              highlightColor: Colors.transparent,
              onTap: () async {

logFirebaseEvent('SETTINGS_PAGE_Icon_hi1op9mo_ON_TAP');
                logFirebaseEvent('Icon_navigate_back');
                try {
                  context.pushNamed(
                    'profile',
                    extra: <String, dynamic>{
                      kTransitionInfoKey: TransitionInfo(
                        hasTransition: true,
                        transitionType: PageTransitionType.fade,
                        duration: Duration(milliseconds: 0),
                      ),
                    },
                  );
                } catch (e) {
                  print(e);
                }
                ;
```

```
            },
            child: Icon(
              Icons.arrow_back_ios_rounded,
              color: Colors.black,
              size: 30.0,
            ),
          ),
        ),
        title: Padding(
          padding: EdgeInsetsDirectional.fromSTEB(7.0, 0.0, 0.0, 0.0),
          child: Text(
            'Settings',
            style: FlutterFlowTheme.of(context).headlineMedium.override(
                  fontFamily: 'Open Sans',
                  color: FlutterFlowTheme.of(context).primaryText,
                  fontSize: 22.0,
                  fontWeight: FontWeight.w500,
                  useGoogleFonts: GoogleFonts.asMap().containsKey(
                      FlutterFlowTheme.of(context).headlineMediumFamily),
                ),
          ),
        ),
        actions: [],
        centerTitle: false,
        elevation: 0.0,
      )
    : null,
  body: Column(
    mainAxisSize: MainAxisSize.max,
    children: [
```

```
Container(
  width: double.infinity,
  decoration: BoxDecoration(
    color: FlutterFlowTheme.of(context).secondaryBackground,
  ),
  child: Padding(
    padding: EdgeInsetsDirectional.fromSTEB(20.0, 0.0, 20.0, 0.0),
    child: Column(
      mainAxisSize: MainAxisSize.max,
      children: [
        Container(
          width: double.infinity,
          height: MediaQuery.sizeOf(context).height * 0.06,
          decoration: BoxDecoration(
            color: FlutterFlowTheme.of(context).secondaryBackground,
          ),
          child: Padding(
            padding:
                EdgeInsetsDirectional.fromSTEB(10.0, 0.0, 0.0, 0.0),
            child: Row(
              mainAxisSize: MainAxisSize.max,
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Row(
                  mainAxisSize: MainAxisSize.max,
                  children: [
                    Icon(
                      Icons.notifications_none,
                      color: Colors.black,
                      size: 27.0,
```

```
    ),
    Padding(
      padding: EdgeInsetsDirectional.fromSTEB(
          28.0, 0.0, 0.0, 0.0),
      child: Text(
        'Notifications',
        style: FlutterFlowTheme.of(context)
            .bodyMedium
            .override(
              fontFamily: 'Open Sans',
              fontSize: 18.0,
              useGoogleFonts: GoogleFonts.asMap()
                  .containsKey(
                      FlutterFlowTheme.of(context)
                          .bodyMediumFamily),
            ),
      ),
    ),
  ],
),
AuthUserStreamWidget(
  builder: (context) => Switch.adaptive(
    value: _model.switchValue1 ??=
        valueOrDefault<bool>(
            currentUserDocument?.receiveNotifications,
            false),
    onChanged: (newValue) async {
      setState(() => _model.switchValue1 = newValue);
      if (newValue) {
        logFirebaseEvent(
```

```
                    'SETTINGS_Switch_yfi5yty8_ON_TOGGLE_ON');
                logFirebaseEvent('Switch_backend_call');

                await currentUserReference!
                    .update(createUsersRecordData(
                  receiveNotifications: true,
                ));
              } else {
                logFirebaseEvent(
                    'SETTINGS_Switch_yfi5yty8_ON_TOGGLE_OFF');
                logFirebaseEvent('Switch_backend_call');

                await currentUserReference!
                    .update(createUsersRecordData(
                  receiveNotifications: false,
                ));
              }
            },
            activeColor: isiOS
                ? FlutterFlowTheme.of(context).mainOrange
                : Colors.white,
            activeTrackColor:
                FlutterFlowTheme.of(context).mainOrange,
            inactiveTrackColor: Color(0xFF909090),
          ),
        ),
      ],
    ),
  ),
),
```

```
Container(
  width: double.infinity,
  height: MediaQuery.sizeOf(context).height * 0.06,
  decoration: BoxDecoration(
    color: FlutterFlowTheme.of(context).secondaryBackground,
  ),
  child: Padding(
    padding:
        EdgeInsetsDirectional.fromSTEB(10.0, 0.0, 0.0, 0.0),
    child: Row(
      mainAxisSize: MainAxisSize.max,
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Row(
          mainAxisSize: MainAxisSize.max,
          children: [
            Icon(
              FFIcons.kphotoonrectangle,
              color: Colors.black,
              size: 24.0,
            ),
            Padding(
              padding: EdgeInsetsDirectional.fromSTEB(
                  30.0, 0.0, 0.0, 0.0),
              child: Text(
                'Photo Library Access',
                style: FlutterFlowTheme.of(context)
                    .bodyMedium
                    .override(
                      fontFamily: 'Open Sans',
```

```
            fontSize: 18.0,
            useGoogleFonts: GoogleFonts.asMap()
                .containsKey(
                    FlutterFlowTheme.of(context)
                        .bodyMediumFamily),
          ),
      ),
    ),
  ],
),
Switch.adaptive(
  value: _model.switchValue2 ??=
      FFAppState().photoLibraryAccess,
  onChanged: (newValue) async {
    if (newValue) {
      logFirebaseEvent(
          'SETTINGS_Switch_vqa186yi_ON_TOGGLE_ON');
      logFirebaseEvent('Switch_update_app_state');

      if (await Permission.storage.status.isDenied) {
        setState(() async {
          try {
            print(await Permission.storage.status);
            await Permission.storage.request();
            if (await Permission
                .storage.status.isGranted) {
              setState(() {
                _model.switchValue2 = newValue;
                FFAppState().photoLibraryAccess =
                    newValue;
```

```
            });
          }
        } catch (e) {
          print(e);
        }
      });
    } else if (await Permission
        .storage.status.isPermanentlyDenied) {
      setPermissionDialog(context);
    }
    if (await Permission.storage.status.isGranted) {
      setState(() {
        _model.switchValue2 = newValue;
        FFAppState().photoLibraryAccess = newValue;
      });
    }
  } else {
    try {
      removePermissionDialog(context);
    } catch (e) {
      print(e);
    }
    if (await Permission.storage.status.isDenied) {
      setState(() {
        _model.switchValue2 = newValue;
        FFAppState().photoLibraryAccess = newValue;
      });
    }
  }
},
```

```
                activeColor: isiOS
                    ? FlutterFlowTheme.of(context).mainOrange
                    : Colors.white,
                activeTrackColor:
                    FlutterFlowTheme.of(context).mainOrange,
                inactiveTrackColor: Color(0xFF909090),
              ),
            ],
          ),
        ),
      ),
    ],
  ),
),
Divider(
  thickness: 10.0,
  color: Color(0xFFEEEEEE),
),
Padding(
  padding: EdgeInsetsDirectional.fromSTEB(15.0, 0.0, 10.0, 0.0),
  child: ListView(
    padding: EdgeInsets.zero,
    shrinkWrap: true,
    scrollDirection: Axis.vertical,
    children: [
      InkWell(
        splashColor: Colors.transparent,
        focusColor: Colors.transparent,
        hoverColor: Colors.transparent,
```

```
                highlightColor: Colors.transparent,
                onTap: () async {

logFirebaseEvent('SETTINGS_PAGE_ListTile_ig1vh2oo_ON_TAP');
                  logFirebaseEvent('ListTile_navigate_to');

                  context.pushNamed('edit_profile');
                },
                child: ListTile(
                  leading: Icon(
                    FFIcons.ksquareandpencil,
                    color: Color(0xFF090F13),
                    size: 22.0,
                  ),
                  title: Text(
                    'Edit Profile',
                    textAlign: TextAlign.start,
                    style:
                        FlutterFlowTheme.of(context).headlineSmall.override(
                              fontFamily: 'Open Sans',
                              fontSize: 18.0,
                              useGoogleFonts: GoogleFonts.asMap().containsKey(
                                  FlutterFlowTheme.of(context)
                                      .headlineSmallFamily),
                            ),
                  ),
                  trailing: Icon(
                    Icons.keyboard_arrow_right_rounded,
                    color: Color(0xFF303030),
                    size: 30.0,
```

```
      ),
      tileColor: FlutterFlowTheme.of(context).secondaryBackground,
      dense: false,
    ),
  ),
  InkWell(
    splashColor: Colors.transparent,
    focusColor: Colors.transparent,
    hoverColor: Colors.transparent,
    highlightColor: Colors.transparent,
    onTap: () async {

logFirebaseEvent('SETTINGS_PAGE_ListTile_fb93dvwg_ON_TAP');
      logFirebaseEvent('ListTile_navigate_to');

      context.pushNamed('report_bug');
    },
    child: ListTile(
      leading: Icon(
        FFIcons.kladybug,
        color: Color(0xFF090F13),
        size: 22.0,
      ),
      title: Text(
        'Log a Bug',
        textAlign: TextAlign.start,
        style:
            FlutterFlowTheme.of(context).headlineSmall.override(
                fontFamily: 'Open Sans',
                fontSize: 18.0,
```

```
                    useGoogleFonts: GoogleFonts.asMap().containsKey(
                        FlutterFlowTheme.of(context)
                            .headlineSmallFamily),
                  ),
            ),
            trailing: Icon(
              Icons.keyboard_arrow_right_rounded,
              color: Color(0xFF303030),
              size: 30.0,
            ),
            tileColor: FlutterFlowTheme.of(context).secondaryBackground,
            dense: false,
          ),
        ),
        InkWell(
          splashColor: Colors.transparent,
          focusColor: Colors.transparent,
          hoverColor: Colors.transparent,
          highlightColor: Colors.transparent,
          onTap: () async {
            await launchURL(

'https://docs.google.com/document/d/1ru9nzc8O7tzBkcFpEnvtJ_6A3G1o9lR
0JQWff6iFxRQ/edit');
          },
          child: ListTile(
            leading: Icon(
              FFIcons.kcheckmarkshield,
              color: Color(0xFF090F13),
              size: 22.0,
```

```
      ),
      title: Text(
        'Privacy Policy',
        textAlign: TextAlign.start,
        style:
            FlutterFlowTheme.of(context).headlineSmall.override(
                fontFamily: 'Open Sans',
                fontSize: 18.0,
                useGoogleFonts: GoogleFonts.asMap().containsKey(
                    FlutterFlowTheme.of(context)
                        .headlineSmallFamily),
              ),
      ),
      trailing: Icon(
        Icons.keyboard_arrow_right_rounded,
        color: Color(0xFF303030),
        size: 30.0,
      ),
      tileColor: FlutterFlowTheme.of(context).secondaryBackground,
      dense: false,
    ),
  ),
  InkWell(
    splashColor: Colors.transparent,
    focusColor: Colors.transparent,
    hoverColor: Colors.transparent,
    highlightColor: Colors.transparent,
    onTap: () async {

logFirebaseEvent('SETTINGS_PAGE_ListTile_bjmy5jmv_ON_TAP');
```

```
  logFirebaseEvent('ListTile_navigate_to');

  context.pushNamed('suggestions');
},
child: ListTile(
  leading: Icon(
    FFIcons.kpencilandoutline,
    color: Color(0xFF090F13),
    size: 22.0,
  ),
  title: Text(
    'Make an App Suggestion',
    textAlign: TextAlign.start,
    style:
        FlutterFlowTheme.of(context).headlineSmall.override(
              fontFamily: 'Open Sans',
              fontSize: 18.0,
              useGoogleFonts: GoogleFonts.asMap().containsKey(
                  FlutterFlowTheme.of(context)
                      .headlineSmallFamily),
            ),
  ),
  trailing: Icon(
    Icons.keyboard_arrow_right_rounded,
    color: Color(0xFF303030),
    size: 30.0,
  ),
  tileColor: FlutterFlowTheme.of(context).secondaryBackground,
  dense: false,
),
```

```
),
InkWell(
  splashColor: Colors.transparent,
  focusColor: Colors.transparent,
  hoverColor: Colors.transparent,
  highlightColor: Colors.transparent,
  onTap: () async {
    context.pushNamed('blocked_users');
  },
  child: ListTile(
    leading: Icon(
      CupertinoIcons.person_crop_circle_fill_badge_exclam,
      color: Color(0xFF090F13),
      size: 22.0,
    ),
    title: Text(
      'Blocked Users',
      textAlign: TextAlign.start,
      style:
          FlutterFlowTheme.of(context).headlineSmall.override(
                fontFamily: 'Open Sans',
                fontSize: 18.0,
                useGoogleFonts: GoogleFonts.asMap().containsKey(
                    FlutterFlowTheme.of(context)
                        .headlineSmallFamily),
              ),
    ),
    trailing: Icon(
      Icons.keyboard_arrow_right_rounded,
      color: Color(0xFF303030),
```

```
            size: 30.0,
          ),
          tileColor: FlutterFlowTheme.of(context).secondaryBackground,
          dense: false,
        ),
      ),
      InkWell(
        splashColor: Colors.transparent,
        focusColor: Colors.transparent,
        hoverColor: Colors.transparent,
        highlightColor: Colors.transparent,
        onTap: () async {

logFirebaseEvent('SETTINGS_PAGE_ListTile_elrl523v_ON_TAP');
          logFirebaseEvent('ListTile_launch_u_r_l');
          await launchURL('http://www.nextmodapp.com/');
        },
        child: ListTile(
          leading: Icon(
            FFIcons.kinfobubble,
            color: Color(0xFF090F13),
            size: 22.0,
          ),
          title: Text(
            'More About Nextmod',
            textAlign: TextAlign.start,
            style:
                FlutterFlowTheme.of(context).headlineSmall.override(
                      fontFamily: 'Open Sans',
                      fontSize: 18.0,
```

```
                    useGoogleFonts: GoogleFonts.asMap().containsKey(
                        FlutterFlowTheme.of(context)
                            .headlineSmallFamily),
                  ),
            ),
            trailing: Icon(
              Icons.keyboard_arrow_right_rounded,
              color: Color(0xFF303030),
              size: 30.0,
            ),
            tileColor: FlutterFlowTheme.of(context).secondaryBackground,
            dense: false,
          ),
        ),
        Builder(
          builder: (context) => InkWell(
            splashColor: Colors.transparent,
            focusColor: Colors.transparent,
            hoverColor: Colors.transparent,
            highlightColor: Colors.transparent,
            onTap: () async {
              logFirebaseEvent(
                  'SETTINGS_PAGE_ListTile_ft0cmivx_ON_TAP');
              logFirebaseEvent('ListTile_share');
              if (Platform.isAndroid) {
                await Share.share(

'https://play.google.com/store/apps/details?id=nextmodapp.com&pcampaignid=web_share',
                    sharePositionOrigin: getWidgetBoundingBox(context),
```

```
      );
    } else if (Platform.isIOS) {
      await Share.share(
        'https://www.nextmodapp.com',
        sharePositionOrigin: getWidgetBoundingBox(context),
      );
    }
  },
  child: ListTile(
    leading: Icon(
      Icons.ios_share,
      color: Color(0xFF090F13),
      size: 24.0,
    ),
    title: Text(
      'Share App',
      textAlign: TextAlign.start,
      style:
          FlutterFlowTheme.of(context).headlineSmall.override(
                fontFamily: 'Open Sans',
                fontSize: 18.0,
                useGoogleFonts: GoogleFonts.asMap()
                    .containsKey(FlutterFlowTheme.of(context)
                        .headlineSmallFamily),
              ),
    ),
    trailing: Icon(
      Icons.keyboard_arrow_right_rounded,
      color: Color(0xFF303030),
      size: 30.0,
```

```
                ),
                tileColor:
                    FlutterFlowTheme.of(context).secondaryBackground,
                dense: false,
              ),
            ),
          ),
          InkWell(
            splashColor: Colors.transparent,
            focusColor: Colors.transparent,
            hoverColor: Colors.transparent,
            highlightColor: Colors.transparent,
            onTap: () async {

logFirebaseEvent('SETTINGS_PAGE_ListTile_7a3833ps_ON_TAP');
              logFirebaseEvent('ListTile_bottom_sheet');
              await showModalBottomSheet(
                isScrollControlled: true,
                backgroundColor: Colors.transparent,
                enableDrag: false,
                context: context,
                builder: (context) {
                  return Padding(
                    padding: MediaQuery.viewInsetsOf(context),
                    child: LogoutWidget(),
                  );
                },
              ).then((value) => setState(() {}));
            },
            child: ListTile(
```

```
    leading: Icon(
      FFIcons.kdoorrighthandopen,
      color: Color(0xFF090F13),
      size: 22.0,
    ),
    title: Text(
      'Logout of Nextmod',
      textAlign: TextAlign.start,
      style:
          FlutterFlowTheme.of(context).headlineSmall.override(
                fontFamily: 'Open Sans',
                fontSize: 18.0,
                useGoogleFonts: GoogleFonts.asMap().containsKey(
                    FlutterFlowTheme.of(context)
                        .headlineSmallFamily),
              ),
    ),
    trailing: Icon(
      Icons.keyboard_arrow_right_rounded,
      color: Color(0xFF303030),
      size: 30.0,
    ),
    tileColor: FlutterFlowTheme.of(context).secondaryBackground,
    dense: false,
  ),
),
InkWell(
  onTap: () async {
    await showModalBottomSheet(
      isScrollControlled: true,
```

```
            backgroundColor: Colors.transparent,
            enableDrag: false,
            context: context,
            builder: (context) {
              return Padding(
                padding: MediaQuery.viewInsetsOf(context),
                child: DeleteUserDialogWidget(),
              );
            },
          ).then((value) => setState(() {}));
        },
        child: ListTile(
          leading: Icon(
            Icons.no_accounts_outlined,
            color: Colors.black,
            size: 22,
          ),
          title: Text(
            'Delete Account',
            style: TextStyle(
                color: Colors.red,
                fontFamily: 'Open Sans',
                fontSize: 18.0),
          ),
        ),
      ),
    ],
  ),
),
```

```
        ],
      ),
    );
  }
}

void showNoMailAppsDialog(BuildContext context) {
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text("Open Mail App"),
        content: Text("No mail apps installed"),
        actions: <Widget>[
          InkWell(
            child: Text("OK"),
            onTap: () {
              Navigator.pop(context);
            },
          )
        ],
      );
    },
  );
}
```

blocked_users_widget.dart

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:google_fonts/google_fonts.dart';
```

```
import 'package:nextmod/auth/firebase_auth/auth_util.dart';
import 'package:nextmod/backend/backend.dart';
import 'package:nextmod/components/loading_widget_widget.dart';
import 'package:nextmod/flutter_flow/flutter_flow_theme.dart';
import 'package:nextmod/flutter_flow/flutter_flow_util.dart';
import 'package:nextmod/settings/blocked_users/blocked_users_model.dart';
import
'package:nextmod/widget_helpers/blocked_user_element/blocked_user_elem
ent_widget.dart';
import 'package:nextmod/widget_helpers/empty_blocked_users.dart';

class BlockedUsersWidget extends StatefulWidget {
  const BlockedUsersWidget({Key? key}) : super(key: key);
  @override
  _BlockedUsersWidget createState() => _BlockedUsersWidget();
}

class _BlockedUsersWidget extends State<BlockedUsersWidget> {
  late BlockedUserModel _model;

  final scaffoldKey = GlobalKey<ScaffoldState>();

  @override
  void initState() {
    super.initState();
    _model = createModel(context, () => BlockedUserModel());
  }

  @override
  void dispose() {
```

```
  _model.dispose();

  super.dispose();
}

@override
Widget build(BuildContext context) {
  Size size = MediaQuery.of(context).size;
  return Scaffold(
    key: scaffoldKey,
    backgroundColor: Colors.white,
    appBar: responsiveVisibility(
      context: context,
    )
        ? AppBar(scrolledUnderElevation: 0,
            systemOverlayStyle: SystemUiOverlayStyle.dark,
            backgroundColor: Color(0x00FFFFFF),
            automaticallyImplyLeading: false,
            leading: Container(
              width: 100.0,
              height: 100.0,
              decoration: BoxDecoration(
                color: FlutterFlowTheme.of(context).secondaryBackground,
              ),
              child: InkWell(
                splashColor: Colors.transparent,
                focusColor: Colors.transparent,
                hoverColor: Colors.transparent,
                highlightColor: Colors.transparent,
                onTap: () async {
```

```
logFirebaseEvent('SETTINGS_PAGE_Icon_hi1op9mo_ON_TAP');
                logFirebaseEvent('Icon_navigate_back');
                context.safePop();
              },
              child: Icon(
                Icons.arrow_back_ios_rounded,
                color: Colors.black,
                size: 30.0,
              ),
            ),
          ),
          title: Padding(
            padding: EdgeInsetsDirectional.fromSTEB(7.0, 0.0, 0.0, 0.0),
            child: Text(
              'Blocked users',
              style: FlutterFlowTheme.of(context).headlineMedium.override(
                    fontFamily: 'Open Sans',
                    color: FlutterFlowTheme.of(context).primaryText,
                    fontSize: 22.0,
                    fontWeight: FontWeight.w500,
                    useGoogleFonts: GoogleFonts.asMap().containsKey(
                        FlutterFlowTheme.of(context).headlineMediumFamily),
                  ),
            ),
          ),
          actions: [],
          centerTitle: false,
          elevation: 0.0,
        )
```

```
    : null,
  body: StreamBuilder<List<UsersRecord>>(
    stream: queryUsersRecord(
      queryBuilder: (usersRecord) => usersRecord.where(
        'blockedBy',
        arrayContains: currentUserReference,
      ),
    ),
    builder: (context, snapshot) {
      if (!snapshot.hasData) {
        return LoadingWidgetWidget();
      }
      ;
      List<UsersRecord> listViewUsersRecordList = snapshot.data!;

      if (listViewUsersRecordList.isEmpty) {
      return  EmptyBlockedUsersWidget();
      }
      return ListView.separated(
        padding: EdgeInsets.zero,
        shrinkWrap: true,
        scrollDirection: Axis.vertical,
        itemCount: listViewUsersRecordList.length,
        separatorBuilder: (_, __) => SizedBox(height: 10.0),
        itemBuilder: (context, listViewIndex) {
          final listViewUsersRecord =
              listViewUsersRecordList[listViewIndex];
          return InkWell(
            splashColor: Colors.transparent,
            focusColor: Colors.transparent,
```

```
                hoverColor: Colors.transparent,
                highlightColor: Colors.transparent,
                onTap: () async {
                  context.pushNamed(
                    'user_profile',
                    queryParameters: {
                      'user': serializeParam(
                        listViewUsersRecord,
                        ParamType.Document,
                      ),
                    }.withoutNulls,
                    extra: <String, dynamic>{
                      'user': listViewUsersRecord,
                    },
                  );
                },
                child: Padding(
                  padding: const EdgeInsets.fromLTRB(20, 15, 20, 0),
                  child: BlockedUserComponentWidget(
                    key: Key(

'Keyunb_${listViewIndex}_of_${listViewUsersRecordList.length}'),
                    user: listViewUsersRecord,
                  ),
                ),
              );
            },
          );
        }),
    );
```

ДОДАТОК Б

ВІДГУК

на кваліфікаційну роботу рівня магістра
«Розробка мобільного додатку для розміщення об'яв для продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter»
студента групи 126м-22-1 Волошина Дмитра Андрійовича.

Метою даної кваліфікаційної работи є розробка мобільного додатку для розміщення об'яв для продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter.

Дана тема актуальна тому, що в контексті швидко ростучого попиту на транспортні засоби в Україні та світі, а відповідно і розвитку ринку та глобалізації, розглядаються питання для створення майданчиків для людей, котрі зацікавлені у збуті або придбанні запчастин та транспортних засобів, або у їх продажі. Люди постійно шукають доступні запчастини для своїх ТЗ, щоб здійснювати ремонт чи покращувати їхній вигляд і функціональність.

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності фахівця галузі знань 12 «Інформаційні технології» спеціальності 126 «Інформаційні системи та технології» – розробка та експлуатація різнопланових інформаційних систем і пов'язаних з ними програмних засобів.

Завдання кваліфікаційної роботи (розробка мобільного додатку для розміщення об'яв для продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter) віднесені в освітньо-професійній програмі підготовки випускника відповідної спеціальності до класу евристичних, рішення яких заснована на знаково-розумових уміннях фахівця.

Оригінальність технологічних рішень полягає в використанні сучасних засобів розробки мобільних додатків та новаторські рішення у створенні програмних алгоритмів та засобів.

Практичне значення результатів кваліфікаційної роботи полягає в створенні інформаційної системи що дозволить людям отримати додаткову платформу для розповсюдження їх об’яв з продажу транспортних засобів та запчастин, та для тих хто зацікавлений у їх придбанні.

Оформлення графічних та текстових матеріалів пояснювальної записки кваліфікаційної роботи виконано на досить високому рівні і без відхилень від стандарту.

Ступінь самостійності виконання кваліфікаційної роботи повною мірою відповідає другому освітньо-кваліфікаційному рівню вищої освіти, тобто ступеню магістра.

В роботі досить повно виконано огляд сучасного стану ринку програмних додатків та сервісів у сфері продажу транспортних засобів та запчастин.

Деякі дискусійні положення та несуттєві недоліки пов’язані з наступними моментами:

– робота дещо перевантажена описом створених алгоритмів та модулів додатку.

– разом з тим не досить чітко викладено та описано структуру роботи, а також недостатньо зрозуміла послідовність виконання розробки фрагменту інформаційної системи;

Незважаючи на вищевказані зауваження, кваліфікаційна робота в цілому заслуговує оцінки

«_____», а її виконавець, студент _____, присвоєння йому кваліфікації магістр за спеціальністю 126 «Інформаційні системи та технології».

Керівник кваліфікаційної роботи,
професор кафедри ІТКІ, В.І. Олєвський

ДОДАТОК В

РЕЦЕНЗІЯ

на кваліфікаційну роботу магістра на тему:
«Розробка мобільного додатку для розміщення об'яв для продажу запчастин та транспортних засобів із гнучким алгоритмом фільтрації на базі Flutter»
студента групи 126м-22-1 Волошина Дмитра Андрійовича.

В контексті швидко зростаючого попиту на транспортні засоби в Україні та світі, розвитку і глобалізації вторинного ринку автомобілів та запасних частин, актуальним стає створення інформаційних систем для учасників цього ринку. Користувачі постійно шукають доступні запчастини для своїх ТЗ, щоб здійснювати ремонт чи покращувати їх вигляд і функціональність і потребують для цього інформаційної підтримки..

В рецензованій кваліфікаційній роботі створено кросплатформений додаток для створення та розміщення об’яв для продажу транспортних засобів та запчастин до них. Додаток дозволяє створювати користувацьку базу і фільтрувати розміщені об’яви за допомогою гнучких алгоритмів відбору. Використані технології розробки та забезпечення функціонування подібних програмних засобів є евристичним компонентом і безпосередньо пов'язане з об'єктом діяльності фахівця спеціальності 126 «Інформаційні системи та технології» галузі знань 12 «Інформаційні технології».

Студент Волошин Дмитро Андрійович досить добре розібрався в специфіці застосування необхідних технологічних засобів для побудови відповідного фрагменту інформаційної системи.

До недоліків роботи слід віднести:

– робота дещо перевантажена описом створених алгоритмів та модулів додатку,

– не досить чітко описано структуру роботи та послідовність виконання розробки фрагменту інформаційної системи.

Однак, розроблений кросплатформений додаток можна вважати закінченим етапом виконаної кваліфікаційної роботи.

На підставі вищевикладеного, можна зробити висновок, що представлені матеріали цілком відповідають вимогам, що пред'являються до кваліфікаційних робіт другого рівня вищої освіти, тобто ступеню магістра.

З огляду на весь спектр створених компонентів, що забезпечують формування даної кваліфікаційної роботи, в цілому вона заслуговує на оцінку «відмінно», а її виконавець, студент Волошин Дмитро Андрійович, присвоєння йому кваліфікації магістр за спеціальністю 126 «Інформаційні системи та технології».

Рецензент, професор кафедри програмного забезпечення комп'ютерних систем, д-р техн. наук **І. С. Лактіонов**