КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня бакалавра

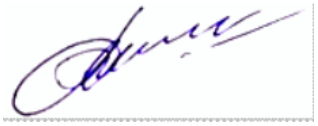
за спеціальністю 122 Комп'ютерні науки
на тему:

МОБІЛЬНИЙ ДОДАТОК ДЛЯ ПРОДАЖУ ЕЛЕКТРОНІКИ У СЕГМЕНТІ РОЗУМНИХ БУДИНКІВ

Виконав студент 4-го курсу
Денис РУДЕНКО


(підпис)

Науковий керівник:
доцент, кандидат технічних наук
Олексій ТКАЧЕНКО


(підпис)

Засвідчую, що в цій роботі немає запозичень
з праць інших авторів без відповідних посилань.
Студент


(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри комп'ютерних наук
« 02 » червня 2023 р.,
протокол № 18
Завідувач кафедри
Микола НІКІТЧЕНКО

Київ - 2023

РЕФЕРАТ

Обсяг роботи: 65 сторінок, 43 ілюстрацій, 13 джерел посилань
UI/UX, ОС, ЗАСТОСУНОК ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID

Об’єктом дослідження даної роботи сфера електронної торгівлі.

Предметом роботи є мобільні додатки для реалізації електронного магазину.

Мета та завдання цієї дипломної роботи є створення програмного мобільного застосунку для продажу електронних девайсів для розумних будинків. Задачі дослідження:

- ознайомлення з існуючими сервісами надання схожих послуг;
- опис програмно-технічних рішень, різних видів забезпечення мобільного додатку;
- ознайомлення з процесом реалізації мобільного додатку від створення прототипу до проведення тестування;
- реалізація на практиці мобільного додатку з усіма видами основного функціоналу

Робота присвячена створенню нативного мобільного застосунку під операційну систему Android із використанням сучасних технологій. В роботу входить розробка клієнтської частини та гарного і зручного UI/UX.

Для розробки оптимізованого мобільного застосунку під ОС Android було проведено дослідження всіх актуальних, на даний час, технологій, що дозволяють реалізувати задачу та роботу з клієнт-серверною частиною. Після досліджень було здійснено порівняння самих популярних компонентів та бібліотек класичної розробки мобільного додатку, їх швидкість, простота реалізації та взаємодії, рівень захищеності, актуальність на різних сегментах ринку та інше.

Вибір технологій для реалізації дипломної роботи був зроблений по цим основним критеріям: масштабованість, наявні можливості, програмне забезпечення та супроводження системи, а також вирішення проблем, які можуть виникнути при розробці нативного мобільного додатку під ОС Android.

У цій роботі приведені результати досліджень та розробки комерційного мобільного додатку, який включає в себе дві версії: користувацьку та для персоналу. Другий варіант додатку дозволяє додавати продукцію в електронний магазин. Робота також містить програмне супроводження, детальні описи встановлення та налаштування розробленого додатку, функції, основні екрани, компоненти, модулі, архітектуру, сервіси та стратегії.

Одним з ключових аспектів розробки було створення гарного та зручного UI/UX, що забезпечує зручність та ефективність взаємодії користувача з додатком. Для цього було використано сучасні техніки та підходи, яку забезпечують консистентність та простоту взаємодії, а також архітектурні підходи, які дозволяють забезпечити розширюваність та підтримку додатку в майбутньому.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ.....5
ВСТУП..... 6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ.9
1.1 Що таке екосистема. Найпопулярніші екосистеми на сьогоднішній час... 10
1.2 Огляд наявних на ринку застосунків..... 12
РОЗДІЛ 2. РОЗРОБКА ЗАСТОСУНКУ ТА ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ.....14
2.1 Вибір інструментальних засобів..... 15
2.2 Розробка UI/UX дизайну та технології для його проектування..... 18
2.3 Технології для навігації. Використані технічні засоби для роботи з екранами.....25
2.4 Розвиток мов програмування під ОС Android.....32
2.5 Технології проектування проекту. Архітектура проекту. Впровадження залежностей.....33
2.6 Технології для роботи з Backend частиною..... 39
2.6.1 Аутентифікація користувача..... 40
2.6.2 Хмарне сховище.....43
2.6.3 База даних у реальному часі..... 46
РОЗДІЛ 3. РОБОТА З СИСТЕМОЮ.....51
3.1 Додаток для користувачів..... 51
3.1.1 Екрани авторизації та реєстрації в додатку..... 51
3.1.2 Екрани магазину, кошику та профілю користувача..... 54
3.2 Додаток для персоналу.....61
ВИСНОВКИ.....63
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

MVVM (Model View ViewModel) - шаблон проектування

NFC (Near Field Communication) - технологія бездротового зв'язку

OS (Operating System) - операційна система застосунку

JIT (Just-In-Time) - динамічна компіляція

JVM (Java Virtual Machine) - віртуальна машина компіляції коду

API (Application Programming Interface) - спосіб комунікації з іншою програмою

HDR (High Dynamic Range) - технологія роботи з зображенням

ML (Machine Learning) - розробка методів для навчання штучного інтелекту

IDE (Integrated Design Environment) - середовище, в котрому розробляються програми

UI (User Interface) - те, як в цілому виглядає інтерфейс

UX (User Experience) - те, як функціонує інтерфейс, чи вдасться у користувача виконати поставлені цілі

XML (eXtensible Markup Language) - мова розмітки

dp (density-independent pixels) - параметр для визначення розміру елементів користувацького інтерфейсу

sp (scale-independent pixels) - параметр для визначення розміру тексту

ВСТУП

[1] У сучасному світі мобільні пристрої стали невід'ємною частиною нашого життя. Вони дозволяють нам залишатися на зв'язку з родиною та друзями, працювати, розважатися та отримувати інформацію. Людина може відправляти електронні листи, проводити відеоконференції та працювати з документами, не виходячи з дому. Це дозволяє нам бути більш продуктивними, ефективними в нашій роботі та навчанні. Усі функції мобільних пристроїв підтверджують їх важливість у нашому житті. Вони стали не просто розвагою, а необхідністю для нас у повсякденному житті.

Не пройшла скрізь і сфера електронної комерції. Кожна людина у сучасному житті колись замовляла та оплачувають щось з мобільного пристрою, а хтось лише таким варіантом наразі і користується, так як зараз це найпростіший та найзручніший спосіб, коли у тебе є все під рукою. З кожним роком все більше бізнесів та компаній переходять до керування і продажів товарів саме з мобільних додатків, особливо це було помітно під час пандемії коронавірусу, коли стрімко зростала сфера обслуговування. Тому було обрано саме цей варіант реалізації електронного магазину для продажу девайсів для розумних будинків.

Мета та завдання даної дипломної роботи є створення програмного мобільного застосунку для продажу електронних девайсів для розумних будинків. Аналіз та впровадження зручного інтерфейсу та розробки функціоналу для редагування запиту купівлі продукції. Завдання дослідження полягають в:

- опис програмно-технічних рішень, різних видів забезпечення мобільного додатку;
- ознайомлення з існуючими сервісами надання схожих послуг;
- ознайомлення з процесом реалізації мобільного додатку від створення прототипу до проведення тестування;
- реалізація на практиці мобільного додатку з усіма видами основного функціоналу

Об’єктом дослідження у цій роботі є сфера електронної торгівлі, а предметом дослідження - програмні засоби розробка мобільного пристрою під мобільну платформу для електронної комерції купівлі та

переглядання гаджетів для розумних будинків, компоненти мобільного додатку, а також програмно-технічні, організаційни засади, принципи, концептуальні підходи до побудови мобільного додатку даної тематики.

Основна важливість теоретичного та практичного значення роботи полягає в тому, що мобільний додаток дозволяє спростити та поліпшити процес продажу електроніки в сегменті розумних будинків. Він надає можливість клієнтам переглядати, обирати та замовляти продукти безпосередньо зі своїх мобільних пристроїв. Це забезпечує зручність та доступність для широкого кола користувачів.

Додаток може включати такі функції, як каталог товарів з докладними описами, фотографіями та цінами, можливість розміщення замовлень з доставкою до дому, оплату онлайн, відстеження статусу замовлення та спілкування з клієнтською службою. Важливо забезпечити зручний і інтуїтивно зрозумілий інтерфейс для користувачів, щоб вони могли легко користуватися додатком та отримувати задоволення від покупок.

Додатково, розробка мобільного додатку для продажу електроніки у сегменті розумних будинків може позитивно вплинути на екологію, зменшивши витрати на транспортування товарів та паперову документацію. Крім того, такий додаток може допомогти клієнтам вибирати більш екологічні пристрої та послуги, що сприятиме збереженню навколишнього середовища.

Розробка мобільного додатку для продажу електроніки у сегменті розумних будинків є важливою та актуальною роботою. У сучасному світі, коли технології швидко розвиваються, розумні будинки стають все більш популярними. Це створює великий попит на електроніку, яка може інтегруватися з домашніми системами автоматизації.

Дослідження показують, що у світі все більше людей користуються розумними технологіями в своєму домі. За даними Statista [2], кількість побутових пристроїв "розумного дому" у світі зростає з кожним роком, передбачається, що кількість підключених пристроїв до 2025 року зросте до 75,44 мільярдів, а споживчі витрати на пристрої, пов’язані з розумним будинком, у всьому світі до 2026 року буде більш ніж 200 мільярдів, цю тенденцію видно на рисунку 1.

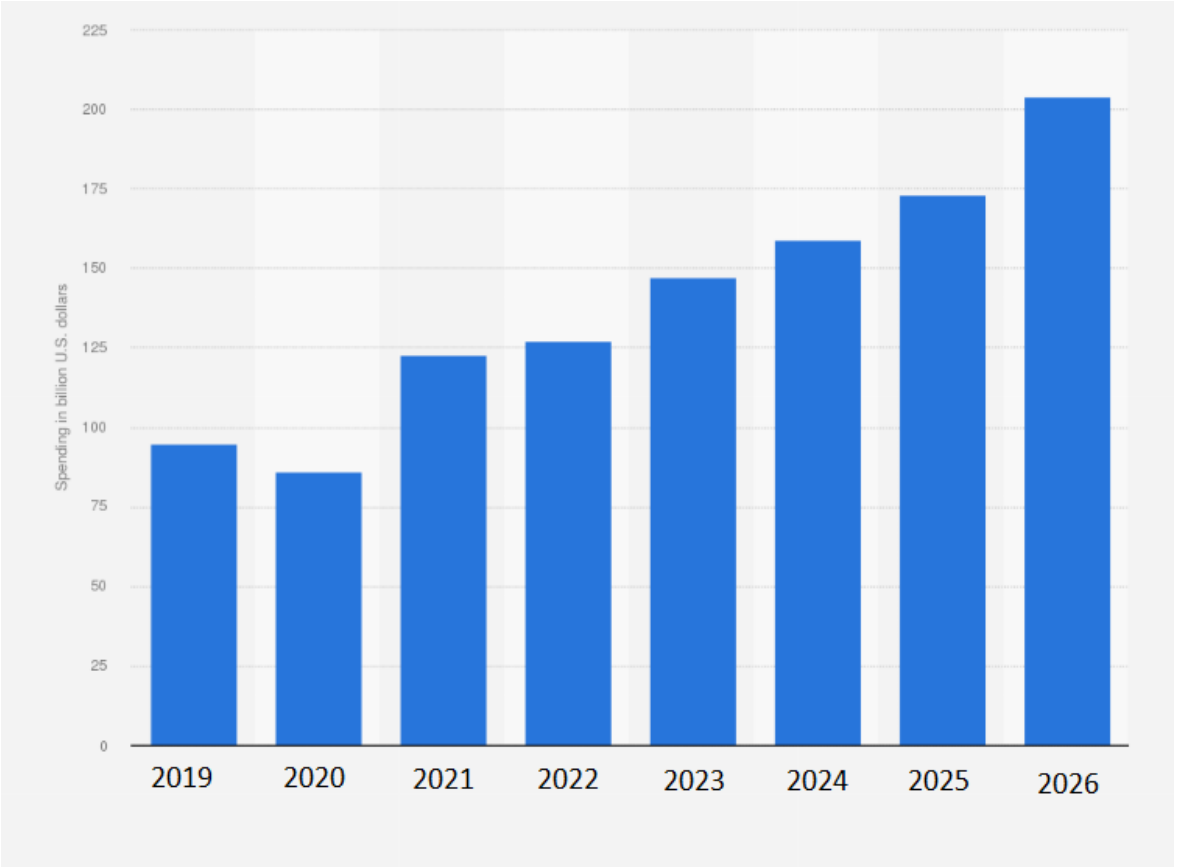


Рисунок 1 - витрати на пристрої для розумних будинків [2]

Так як конкуренція наразі на цьому ринку дуже мала, то розробка мобільного додатку для продажу електроніки в сегменті розумних будинків є дуже важливою для бізнесу, який бажає бути конкурентоспроможним на цьому ринку.

Отже, розробка мобільного додатку для продажу електроніки у сегменті розумних будинків є важливою і перспективною задачею, яка має великий потенціал для розвитку бізнесу та позитивного впливу на екологію.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

У сучасному світі все більшої популярності набувають розумні будинки, які дають можливість забезпечити комфортне проживання за допомогою технологічних інновацій. Одним із ключових елементів таких будинків є електроніка, яка все більше і більше інтегрується в різноманітні системи.

[3] Розумний дім – це інноваційний продукт, який включає в себе різні системи. За допомогою різних датчиків і систем управління розумні будинки можуть контролювати температуру, освітлення, безпеку та інші функції, що робить їх більш привабливими для споживачів. Однією з найважливіших систем є система «розумних» пристроїв, яка включає різноманітну електроніку – від освітлення до систем безпеки.

У кваліфікаційній роботі був розроблений мобільний додаток для продажу цієї самої електроніки для розумних будинків. Таким чином, користувачеві не потрібно кудись йти, щоб ознайомитися з пристроями в якомусь офлайн магазині, адже в нашому додатку всі товари, які можуть зацікавити покупця, мають детальний опис, зручний інтерфейс і необхідні креслення для продукт. Весь функціонал доступний відразу після реєстрації в додатку і взаємопов'язаний, що не відволікає користувача від його завдання. Цей додаток отримав назву «TranquilHaven», і це саме той момент, коли назва говорить сама за себе — безпроблемна покупка електронних пристроїв для розумних будинків, щоб побудувати вдома свій окремий світ.

Особливістю таких будівель є те, що всі системи забезпечуються єдиним центром управління, яким зручно керувати з мобільного пристрою, цей єдиний центр управління називається «екосистема».

Наразі вже достатньо компаній, які створюють пристрої та надають свою екосистему для зручного використання.

1.1 Що таке екосистема. Найпопулярніші екосистеми на сьогоднішній час.

[3] Екосистеми для смарт-будинків - це інтегровані платформи та пристрої, призначені для автоматизації та керування різними аспектами домашнього середовища. Вони об'єднують різні пристрої та системи в єдину систему управління, дозволяючи користувачам контролювати освітлення, опалення, кондиціювання повітря, безпеку, розваги та інші аспекти будинку за допомогою зручного інтерфейсу, такого як мобільний додаток або голосовий помічник.

Екосистеми для смарт-будинків зазвичай будуються навколо центрального пристрою, відомого як "хаб" або центр управління. Хаб є мозком системи та забезпечує зв'язок між різними пристроями та додатками. Він може використовувати різні технології зв'язку, такі як Wi-Fi, Bluetooth та інші.

В додатку дипломної роботи на даний час впроваджено та продаються девайси чотирьох екосистем, які є популярними розумними помічниками для дому, які доступні на ринку. Пристрої цих компаній можуть керувати різними системами дому, включаючи освітлення, опалення та безпеку, що робить їх більше зрічними для споживачів. Екосистеми які входять до цього списку:

- **Amazon Alexa**

Amazon Alexa — це хмарна голосова служба, яка працює на розумних колонках і моніторах Echo та Echo Dot. Він може відповідати на запитання, відтворювати музику та керувати всіма пристроями розумного будинку. Завдяки навичкам сторонніх розробників Alexa також може замовляти її, забезпечувати поїздками та навіть керувати автомобілем.

- **Google Home**

Від розумного світильника до розумного замка Google Home об’єднує пристрої та служби від Google і ваших улюблених брендів, щоб створити персоналізований розумний дім, який допоможе вам виконувати справи. За допомогою док-станції ви можете відтворювати музику, відповідати на запитання та керувати різними пристроями розумного дому. За допомогою Google Assistant користувачі також можуть дзвонити та надсилати повідомлення.

- **Apple HomeKit**

Apple HomeKit — це розумна домашня платформа, яка дозволяє користувачам керувати різноманітними розумними домашніми пристроями зі свого iPhone або iPad. Він підтримує широкий спектр пристроїв, включаючи освітлення, термостати та замки. За допомогою Siri користувачі також можуть керувати своїми розумними домашніми пристроями за допомогою голосових команд.

- **Samsung SmartThings**

Samsung SmartThings — це платформа розумного дому, яка дозволяє користувачам зручно керувати різними пристроями розумного дому зі свого смартфона. Це дозволяє вам керувати освітленням, температурою, безпекою та іншими аспектами вашого дому з одного місця. Завдяки цьому ви зможете не тільки заощадити час, а й підвищити комфорт свого життя.

Крім того, Samsung Smart Things забезпечує сумісність з великою кількістю пристроїв розумного дому від самої компанії Samsung Electronic, що дозволяє користувачам вибирати найкращі рішення для свого дому. Користувач може підключити до неї різні пристрої, такі як термостати, дверні замки, камери безпеки, мультимедійні системи та багато іншого.

Samsung SmartThings дозволяє користувачам створювати розумні сценарії, що дозволяє розширити можливості вашого розумного дому. Користувач може налаштувати різні сценарії, щоб вони автоматично включали різні пристрої в залежності від певних умов. Наприклад, може налаштувати сценарій, щоб він автоматично включав світло, коли власник повертається додому пізно ввечері. Скористувавшись повною екосистемою Samsung та його платформою розумних девайсів SmartThings можно затвердити, що завдяки Samsung SmartThings користувач зможете насолоджуватись зручністю та комфортом розумного дому у повній мірі.

1.2 Огляд наявних на ринку застосунків

Кожен з девайсів екосистеми можно купити на сайті виробника. Якщо це HomeKit, то в Apple Store, якщо SmartThings - Samsung Shop та інші, але при покупці або заказу можуть виникнути великі проблеми які утруднить купівлю цих самих девайсів, наприклад, при покупці на сайті виробника можуть виникнути що не зрозуміло буде статус замовлення.

Саме за цих проблем виникає необхідність в інших застосунках, де ми можемо спокійно придбати електронні девайси. Тем не менш, це зручніше ніж від самого виробника.

Проаналізувавши комерційний ринок продажу девайсів для розумних будинків, було знайдено деякі варіанти:

1. На сайті виробника (Amazon Alex, Apple Store, інші)
2. Інтернет-магазини такі як “Rozetka”, “АЛЛО”, “Comfy” тощо
3. Інтернет-магазин розумних пристроїв для дому “HomeHub”
4. Інтернет-магазин розумних пристроїв для дому “WiseHome”

Якщо з першим варіантом ми розібралися, то перейдемо до наступних трьох

Магазини як “Rozetka”, “Алло” та інші спеціалізуються на техніці і в них є також асортимент для розумних будинків, але хоча електронні магазини можуть мати певний асортимент пристроїв для розумного будинку, вони не завжди пропонують повний асортимент продуктів, доступних на ринку. Деякі нові версії або спеціалізовані пристрої можуть бути відсутніми у їхньому асортименті, так як вони можуть рахувати це за непотрібне, як наприклад з деякими девайсами Amazon.

Також, так як політика деяких цих магазинів змінилась, шанс ризику попасти на підробку стрімко зросло, так як в цих магазинах може бути продавець не сам наприклад “Rozetka”, а інші продавці котрі отримали ліцензію продавати свої товари на цьому сайті. Тому в електронних магазинах існує певний ризик придбання підроблених або неякісних пристроїв, це теж саме що прибрати девайс для розумного будинку на сайтах із онлайн оголошеннями.

Перейдемо до інтернет-магазину “HomeHub”. Проаналізувавши цей інтернет-магазин, було помічено що ця компанія немає мобільного застосунку, що вже значний плюс до нашого проекту, так як користувач, який бажає замовити електронний девайс в будь-якому місці в будь-який час, не зможе цього зробити. Також проаналізувавши їх ринок, було

помічено, що хоч в них написано що вони продають електронні девайси під HomeKit чи SmartThings, окрім якийсь китайских виробників товарі у наявності немає, що означає проблеми з доставкою чи взагалі ці девайси були прибрані.

Переходячи до останнього інтернет-магазину “WiseHome”, було помічено, що цей магазин також не має свого мобільного додатку, але в них вже асортимент товарів більше. Але подивившись та проаналізувавши цей додаток, можно побачити, що навіть якщо ти обираєш щоб подивитись асортимент, наприклад, девайсів від Google, користувачу одразу ж показується невідомі китайські виробники, девайси котрих хоч і можуть підключатися до Google Home, але про якість цих пристроїв можно засумніватися. Тому коли користувач забажає прибрати товар від відомого виробника, девайси від нього він не знайде.

Це все каже про те, що на ринку програмного забезпечення для смартфонів для операційної системи Android та взагалі схожих інтернет-магазинів бракує, для купівлі електроніки для розумних будівель.

РОЗДІЛ 2. РОЗРОБКА ЗАСТОСУНКУ ТА ОГЛЯД ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Зараз мобільні платформи займають одне з провідних місць на ринку електроніки. Це призвело до того, що розробка додатків під мобільні пристрої стала однією з найбільш важливих галузей інформаційної технології.

Одним з ключових аспектів мобільної розробки є графіка та сама розробка користувацького інтерфейсу. Графіка в мобільних додатках може використовуватися для покращення візуального досвіду користувачів, а також для відтворення відео та ігор. Для розробки графіки в мобільних додатках можна використовувати різні інструменти, такі як OpenGL ES, Unity та Unreal Engine. OpenGL ES є стандартом для графічного програмування на мобільних пристроях, тоді як Unity та Unreal Engine є популярними інструментами для розробки ігор на мобільних платформах.

[4] Операційні системи для мобільних пристроїв постійно розвиваються та оновлюються. Розробники намагаються поліпшити функціональність та продуктивність систем, а також забезпечити безпеку користувачів та їхніх даних. Крім того, з'являється все більше технологій, таких як штучний інтелект та розпізнавання голосу, які можуть бути використані в операційних системах для поліпшення їхньої функціональності.

Наразі найпопулярнішими операційними системами для мобільних пристроїв є Android та iOS. Та наразі світ мобільних додатків в основному розділений на цих двох операційних систем. За даними Statista, ці дві операційні системи займають приблизно 98% світового ринку додатків. У той час як iOS займає понад 27% світового ринку, на частку Android припадає понад 71%, що дає їй величезну перевагу в охопленні ширшої аудиторії [5]. Ця тенденція за 2010-2020 роки зображена на рисунку 2.

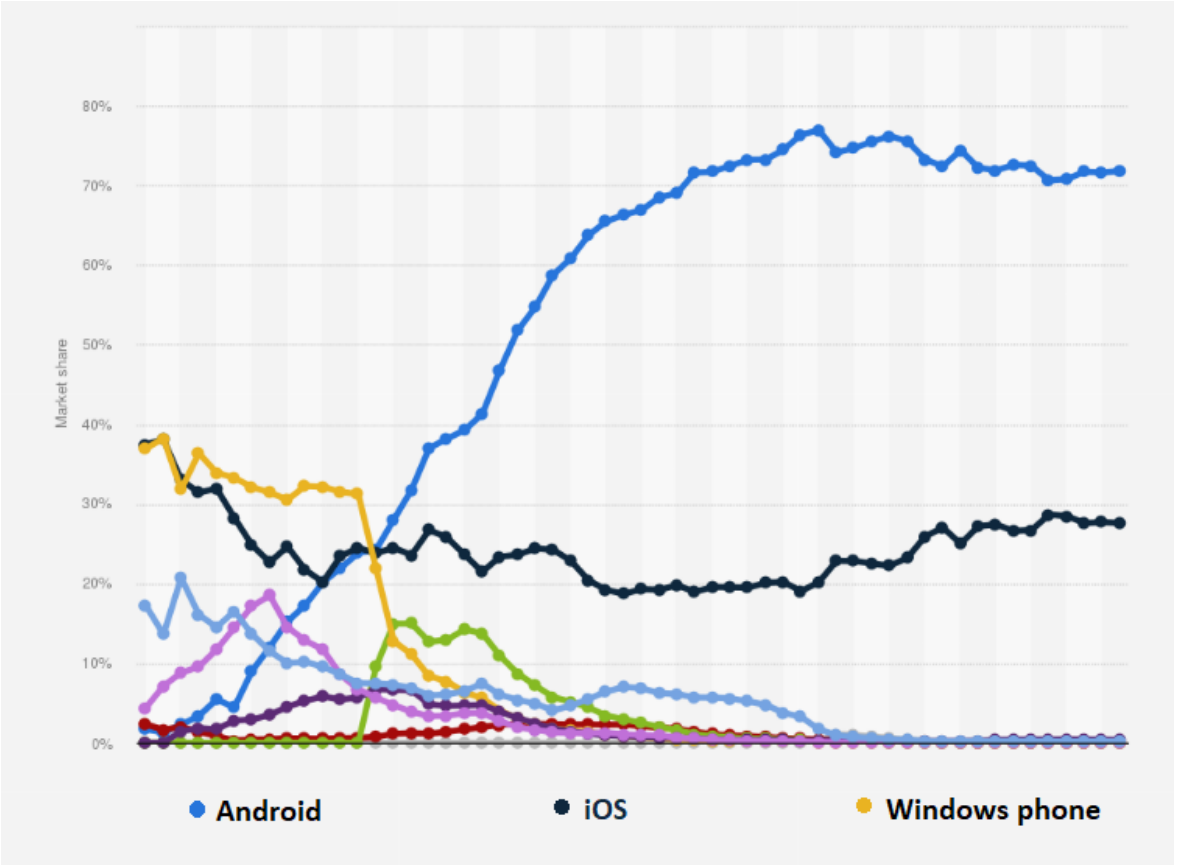


Рисунок 2 - популярність трьох OS під мобільні платформи

В цій дипломній роботі було використовувати саме операційну систему Android. Android — це відкрита платформа від Google, яка надає користувачам велику кількість можливостей. Його можна встановити на різні пристрої, від недорогих до флагманських. Програми для Android можна розробляти різними мовами програмування, такими як Java, Kotlin, C++ тощо. Android підтримує багатофункціональний екран, який дозволяє користувачам використовувати пристрій для багатьох цілей, включаючи роботу, розваги та спілкування.

2.1 Вибір інструментальних засобів

В цьому пункті буде розглянуто обрані засоби для розробки мобільного додатку такі як IDE, системи контролю версій, збирачі проектів. Без цих інструментів при сучасній розробці не може обійтись жоден з розробників, тому важливо виділити цьому увагу.

Android Studio — це інтегроване середовище розробки (IDE) для розробки програмного забезпечення для платформи Android. Він надає

розробникам усі інструменти, необхідні для створення, тестування, налагодження та розгортання програм Android.

Середовищем програмування було обрано Android Studio, оскільки на ринку немає кращої альтернативи для розробки Android-додатків. Він має функцію встановлення емуляторів і фізичних пристроїв. Android Studio поставляється з емулятором Android, який дозволяє розробникам емулювати різні пристрої Android для тестування програм. Також підтримується підключення фізичних пристроїв для тестування та налагодження. Тому для розробки використовувалося саме це середовище програмування і вся робота виконувалася в ньому.

Переходимо до системи контролю версій. Для роботи використовувалася система управління Git, оскільки це найпоширеніша і зрозуміла технологія.

[6] Git - це розподілена система керування версіями, яка використовується для відстеження змін у файловій системі та спільної роботи над проектами програмного забезпечення. Вона була розроблена Лінусом Торвальдсом з метою керування розробкою ядра Linux, але зараз використовується в різних проектах по всьому світу.

[7] Особливу увагу приділено саме системі збирання проектів. Найпоширеніші з них між Java та Kotlin розробниками є дві збирачі проектів: Gradle та Maven.

- Синтаксис і конфігурація

Maven використовує XML для конфігурації проекту та залежностей, тоді як Gradle використовує Groovy або Kotlin DSL (Domain-Specific Language). DSL-підхід у Gradle надає більш гнучкий і зрозумілий синтаксис, що полегшує налаштування проекту.

- Продуктивність та продвинуті можливості

Gradle має свою високу продуктивність за рахунок інкрементальних збірок, які забезпечують швидке виконання лише однієї задачі. Він також підтримує більші продвинуті можливості, такі як скриптовані завдання, конфігураційні блоки на рівнях проекту та модуля, та можливість використовувати плагіни, написані на різних мовах програмування.

- Легкість використання

Gradle має простіший та інтуїтивніший синтаксис порівняно з Maven, особливо для розробників, які знайомі з мовами програмування,

такими як Groovy або Kotlin. Він дозволяє швидко налаштовувати та змінювати проекти, що полегшує роботу.

Не дивлячись на те, що може здатися що майже немає різниці яку збірку проекту обрати, в мобільній розробці під операційну систему Android використовують лише саме Gradle. Це тому що Gradle був офіційно прийнятий Google як система збирання для проектів Android. Він має вбудовану підтримку Android SDK та дозволяє легко налаштувати залежності, ресурси та збірку додатків Android.

Інкрементальна збірка та ефективне кешування залежностей в Gradle дозволяють прискорити час збірки проекту, що є важливим у мобільній розробці, де час розробки може бути обмеженим. Тому швидкість збірки саме з використанням Gradle краще.

Gradle надає розробникам більшу гнучкість при конфігурації проектів та збірці. Він дозволяє легко виконувати специфічні для мобільної розробки завдання, такі як генерація ресурсів, підписування додатків і розгортання на пристрої. Файли Gradle розбиваються на 2 види, для модулів (під-проекти) та для самого проекту. Основний вигляд файлу Gradle для проекту зображен на рисунку 3.

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    dependencies {
        classpath "com.android.tools.build:gradle:7.0.3"
        classpath 'org.jetbrains.kotlin:kotlin-gradle-plugin:1.6.10'
        classpath 'com.google.gms:google-services:4.3.14'
        //safe args
        def nav_version :String = "2.4.1"
        classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
        classpath 'com.google.dagger:hilt-android-gradle-plugin:2.43.2'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}
plugins {
    id 'com.android.application' version '7.2.2' apply false
    id 'com.android.library' version '7.2.2' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.10' apply false
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Рисунок 3 - Головний файл Gradle для залежностей

Враховуючи ці фактори, Gradle є популярним вибором для мобільної розробки на платформі Android. Однак, Maven також залишається широко використовуваним в інших областях програмування

та має свої переваги і його можуть використовувати в інших сферах розробки як веб чи блокчейн.

2.2 Розробка UI/UX дизайну та технології для його проектування

Важливо, для того, щоб користувач залишився користуватися мобільним додатком потрібно не тільки зробити функціональну частину, але і гарну інтерфейсну.

UI/UX - це поняття, яке відноситься до процесу розробки та дизайну веб-сайтів, мобільних додатків та інших програмних продуктів, щоб забезпечити користувачам максимально комфортне та ефективне взаємодію з ними. Розділимо їх та розпишемо окремо:

- Користувацький інтерфейс (UI) - це те, що бачить користувач на екрані свого комп'ютера, смартфона або планшета під час взаємодії з програмним продуктом. Це може бути меню, кнопки, форми, вікна та інші елементи інтерфейсу. Основна мета UI - зробити інтерфейс зручним та простим у використанні для користувача.
- Користувацький досвід (UX) Користувацький досвід - це враження, яке користувач отримує від взаємодії з програмним продуктом. UX охоплює весь процес, від першого відкриття програми до закриття. Ефективний UX має забезпечити користувачеві легкий доступ до необхідної інформації та функцій, а також допомогти йому зрозуміти, як ці функції працюють.

Розробка інтерфейсу користувача (UI) для мобільних пристроїв має свої особливості та аспекти, які слід враховувати. Ось деякі ключові аспекти дизайну мобільного інтерфейсу користувача:

1. Адаптивний дизайн: враховує різні розміри екрана та орієнтації мобільних пристроїв. Програма має добре виглядати та працювати на різних пристроях, включаючи смартфони, планшети та навіть розумні годинники. Адаптивний дизайн дозволяє оптимізувати розміщення елементів, шрифтів та інших параметрів для забезпечення комфортної взаємодії з додатком незалежно від розміру екрана.
2. Ергономіка та простота використання: мобільні програми мають бути простими у використанні та інтуїтивно зрозумілими. Важливо максимально спростити навігацію, забезпечити зручний доступ до

основних функцій і скоротити кількість кроків для досягнення бажаного результату. Принципи матеріального дизайну Google для операційної системи Android і Рекомендації щодо людського інтерфейсу Apple для операційної системи iOS містять вказівки щодо створення зручного та простого інтерфейсу.

3. Візуальний дизайн: візуальний аспект мобільного додатка дуже багатий, щоб створити привабливий і сучасний вигляд. Використання правильних кольорів, типографіки, піктограм і графіки створює привабливий дизайн, який відповідає бренду та привабливий.
4. Взаємодії та анімація: мобільні пристрої підтримують різні способи взаємодії, зокрема дотики, жести, змахування тощо. Використання цих функцій дозволяє створити більш інтуїтивно зрозумілу та привабливу взаємодію з користувачем. Додавання ефектів анімації може покращити взаємодію та зробити програму більш живою та привабливою.

Підсумовуючи, UI/UX є важливим елементом розробки програмного забезпечення та веб-дизайну. Забезпечення зручного та ефективного користування може мати значний вплив на успіх продукту та залучення користувачів. Тому дизайнери та розробники повинні приділяти достатню увагу UI/UX при розробці програмного продукту.

Однією з найбільших проблем у розробці UI/UX для Android є проблема фрагментації. Android має велику кількість пристроїв з різними розмірами екрана, роздільною здатністю та співвідношенням сторін. Це ускладнює створення адаптивного та узгодженого UI/UX на всіх пристроях. Розробники мають переконатися, що їхній UI/UX дизайн сумісний із широким спектром пристроїв, що може забрати багато часу та дорого.

Навігація є невід’ємною частиною дизайну інтерфейсу користувача та UX, і її може бути складно використовувати на Android. На відміну від iOS, Android має кнопку «Назад», що іноді може призвести до заплутаних шляхів навігації. Розробники повинні забезпечити інтуїтивно зрозумілий і простий у використанні дизайн навігації, а також врахувати функції кнопки «Назад».

Для розробки користувацького інтерфейсу під Android додатки використовуються XML Layouts. Замість використання редактора макета, можно створити макет своєї програми, змінивши XML, який

описує інтерфейс користувача (рисунок 4). Для розробник Android важливо навчитися розуміти та змінювати макети інтерфейсу користувача за допомогою XML.

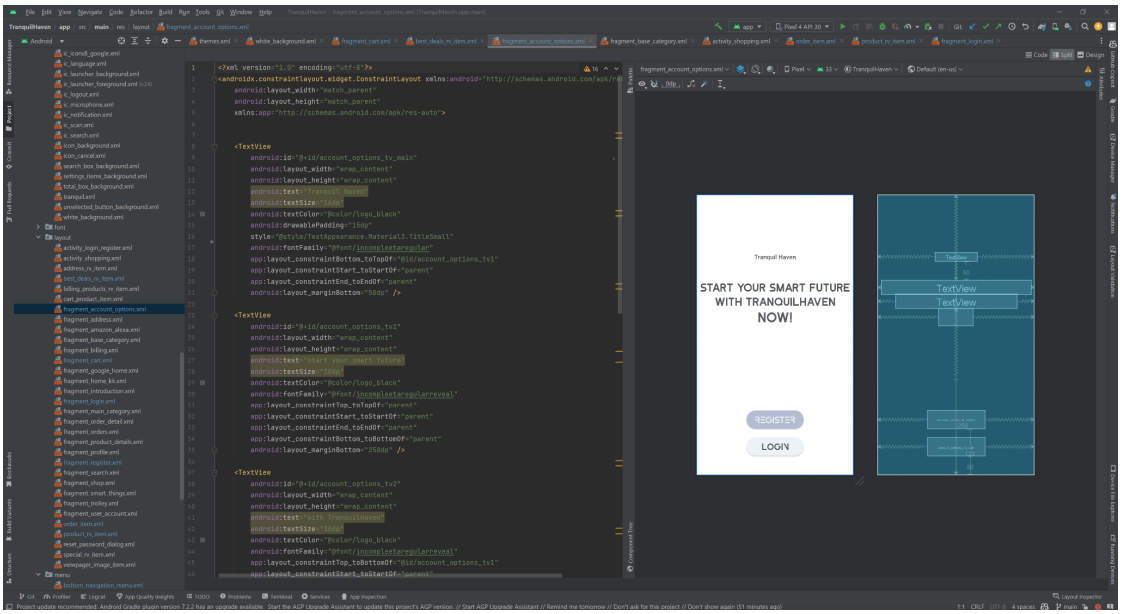


Рисунок 4 - Макет та редактор користувацького інтерфейсу

Розробники переглядають та редагують XML-файл, який визначає макет інтерфейсу користувача для цієї програми. XML розшифровується як eXtensible Markup Language, який є способом опису даних за допомогою текстового документа. Оскільки XML є розширюваним і дуже гнучким, він використовується для багатьох різних речей, зокрема для визначення макета інтерфейсу користувача програм Android

Щоб задати інтерфейс користувачу в мобільній Android розробці, треба написати компоненти Layouts та Views.

[9] Layout - це механізм, який використовується в Android для розміщення і керування елементами інтерфейсу користувача на екрані. Layout-компоненти дозволяють розміщувати елементи в різних комбінаціях та забезпечують більш гнучкий та точний дизайн користувацького інтерфейсу Android-додатків.

У розробці Android View - це об’єкт, який малює щось на екрані, з яким користувач може взаємодіяти. Видами можуть бути кнопки, текстові поля, зображення та багато інших типів елементів інтерфейсу користувача. Вони є будівельними блоками інтерфейсу користувача в програмі Android.

Кожен перегляд є екземпляром класу, який розширює клас `View` або один із його підкласів у Android SDK. Перегляди можна додавати до

макета, який є контейнером, який впорядковує види на екрані певним чином.

Розробник описуєте ієрархію перегляду елементів інтерфейсу користувача на екрані. Наприклад, ConstraintLayout (батьківський) може містити кнопки, TextViews, ImageViews або інші види (дочірні). Головне пам’ятати, що ConstraintLayout є підкласом ViewGroup. Це дозволяє гнучко розташовувати або розмірювати дочірні види (рисунок 5).

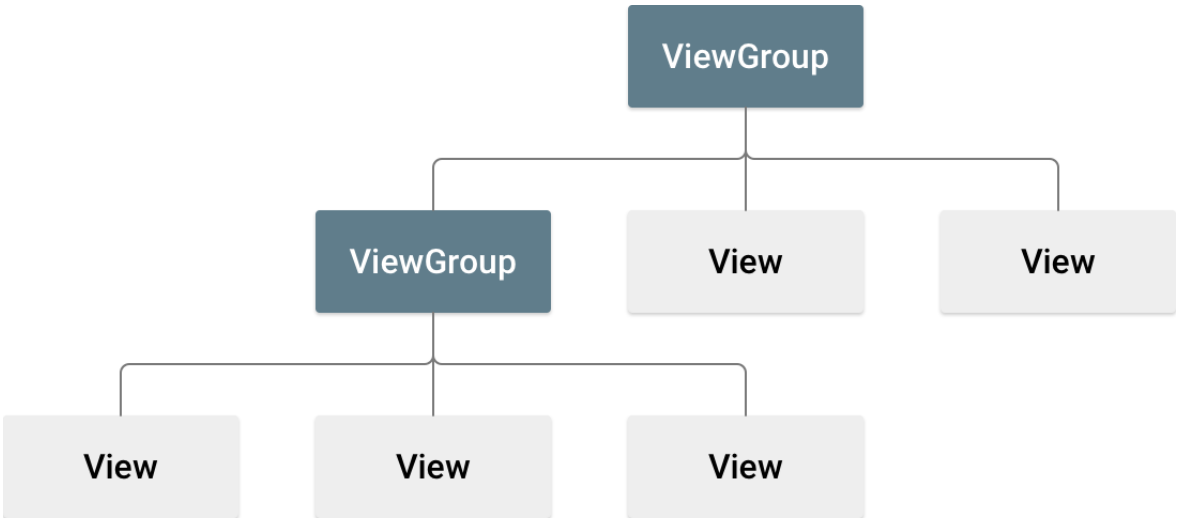


Рисунок 5 - Ієрархія вмісту програми Android [9]

Кожен елемент інтерфейсу користувача представлено елементом XML у файлі XML. Кожен елемент починається і закінчується тегом, і кожен тег починається з < і закінчується на >. Подібно до того, як ви можете встановити атрибути для елементів інтерфейсу користувача за допомогою редактора макета (дизайн), елементи XML також можуть мати атрибути. Візьмемо приклад з цієї роботи та покажемо як спрощено XML для елементів інтерфейсу може виглядати (рисунок 6):

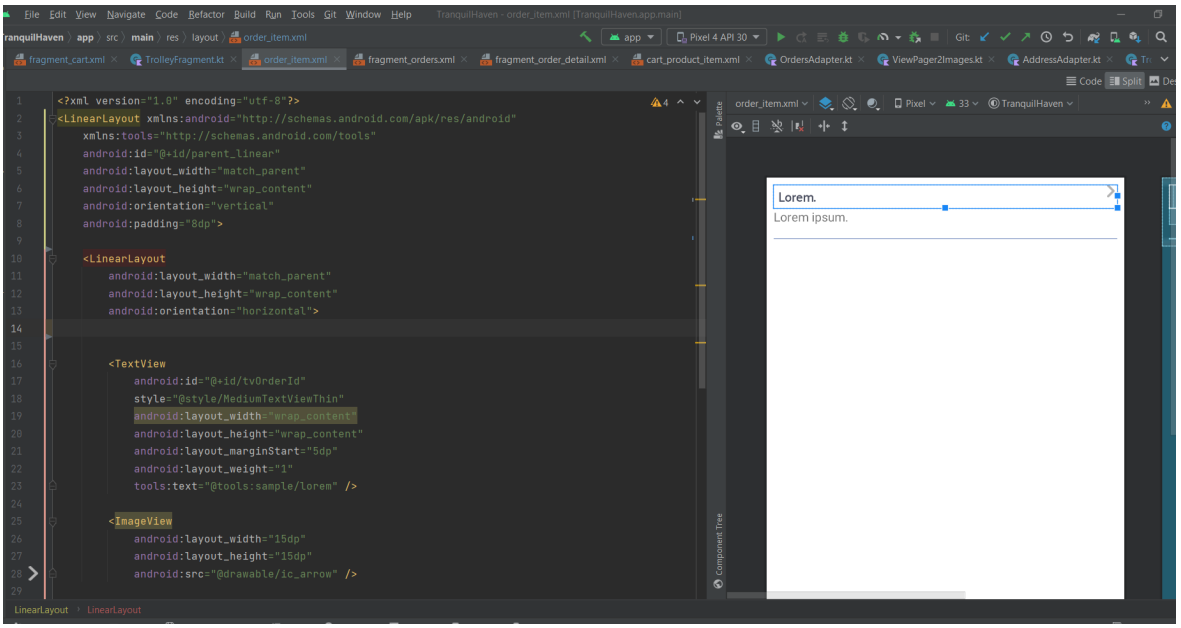


Рисунок 6 - Ієрархія елементів інтерфейсу

Кожен з Layouts та Views має містити в собі атрибути розміру висоті android:layout_height та ширини android:layout_width. Задамо розмір в атрибутах ширини та висоті в dp, але для тексту закладено так, що потрібно писати розмір в sp.

Для того щоб зробити анімовані кнопки, було використано бібліотеку“LoadingButtonAndroid”, при натисканні користувач побачить гарну анімацію та сама кнопка буде з охайними краями. Для того щоб додати цю бібліотеку, потрібно в build Gradle додати в блоці “dependency” одну стрічку:
implementation("com.github.leandroborgesferreira:loading-button-android:2.3.0")

Для плавного користування з мобільним застосунком було зроблено пару анімацій, створивши в папці з ресурсами “res” нову папку з анімаціями. Виглядає анімація так:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<translate android:fromYDelta="100%"
android:toYDelta="0%"
android:duration="250" />
</set>
```

В цьому XML-файлі використовується елемент <set>, який представляє собою контейнер для групи анімацій. Потім всередині <set>

знаходиться елемент <translate>, який описує конкретну анімацію перекладу.

В атрибуті xmlns:android виявляється простір імені Android для елементів XML.

Атрибути android:fromYDelta і android:toYDelta визначають початкове і кінцеве положення елемента по вертикалі. Значення 100% в android:fromYDelta означає, що елемент ізначально знаходиться внизу екрану (100% від висоти екрану), а значення 0% в android:toYDelta означає, що елемент має бути переміщений у верхній екран (0% від висоти екрану).

Атрибут android:duration визначає тривалість анімації в мілісекундах. У цьому випадку анімація буде займати 250 мілісекунд.

Таким чином, даний код визначає анімацію, при якій елемент буде поступово переміщатися внизу екрану вгору за 250 мілісекунд.

Для того щоб зробити якісний та гарний додаток, треба дотримуватись стилістики гарного написання користувацького інтерфейсу. В Android розробці звичайно використовують для цього стилістику від Material Design, для цього треба додати їх бібліотеку:
implementation 'com.google.android.material:material:1.7.0'

Material Design — це стиль дизайну, розроблений Google і використовується в програмах та продуктах, зокрема в Android, Google Docs та багатьох інших. Він визначає, як мають виглядати елементи інтерфейсу користувача, як користувач має з ними взаємодіяти та як вони повинні поводитися в різних ситуаціях.

За допомогою цих інструментів було зроблено основний екран додатку. За цим можно спостерігати на головному екрані користувача, коли він відкрив додаток (рисунок 7)

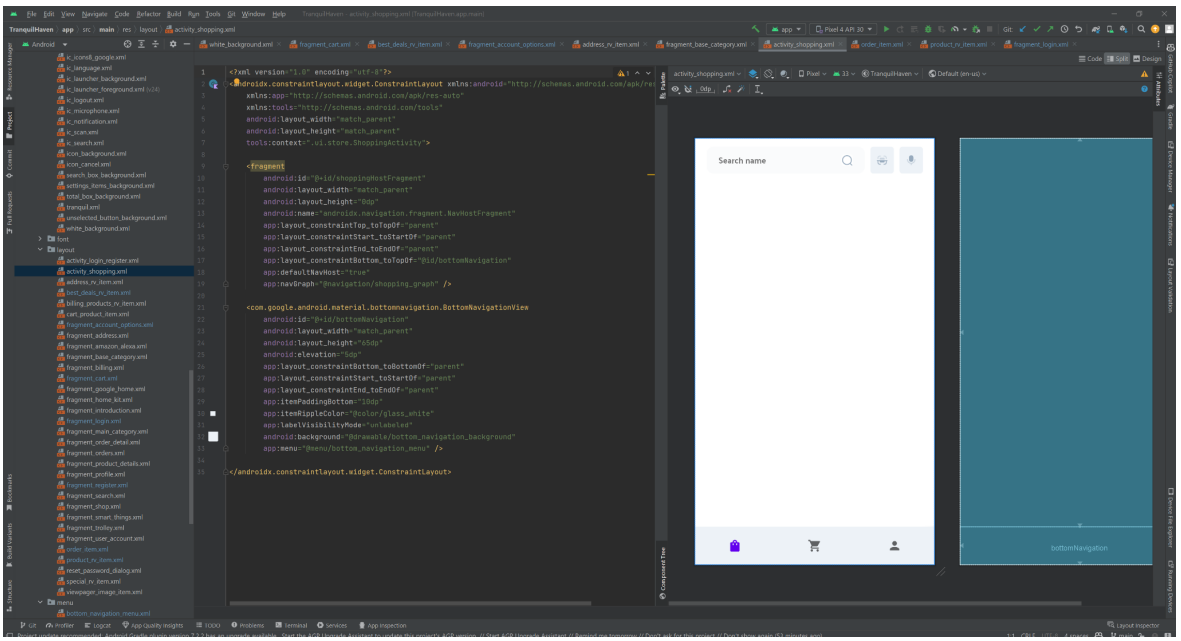


Рисунок 7 - Головний екран користувача

Для додавання основного інтерфейсу користувача для головного екрану, створено шаблон, де користувач може розглядати основні списки товарів, чи обрані товари за обраною компанією (рисунок 8).

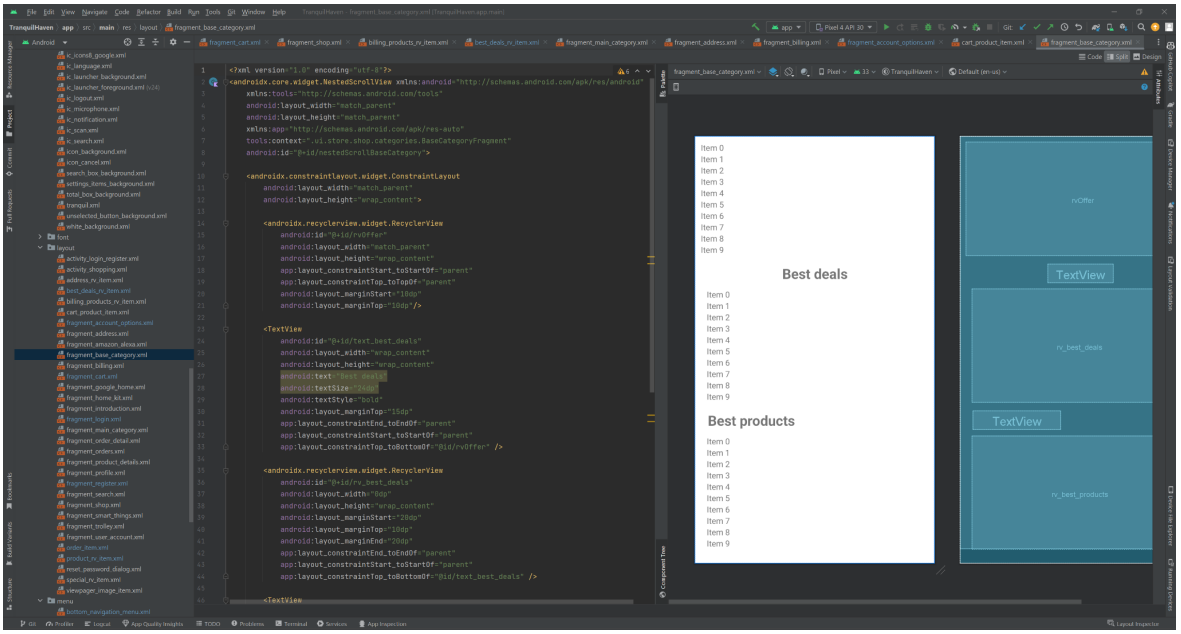


Рисунок 8 - Шаблон по якому будується основний інтерфейс користувача в магазині

Самі списки створені за допомогою бібліотеки RecyclerView. [10] RecyclerView дозволяє легко ефективно відображати великий набір даних. Враховуючи дані та визначаючи, як виглядає кожен елемент,

бібліотека RecyclerView динамічно створює елементи, коли вони потрібні.

Щоб задати елементи нашому списку, нам потрібно для нього також створити екран, але з меншим розміром (рисунок 9).

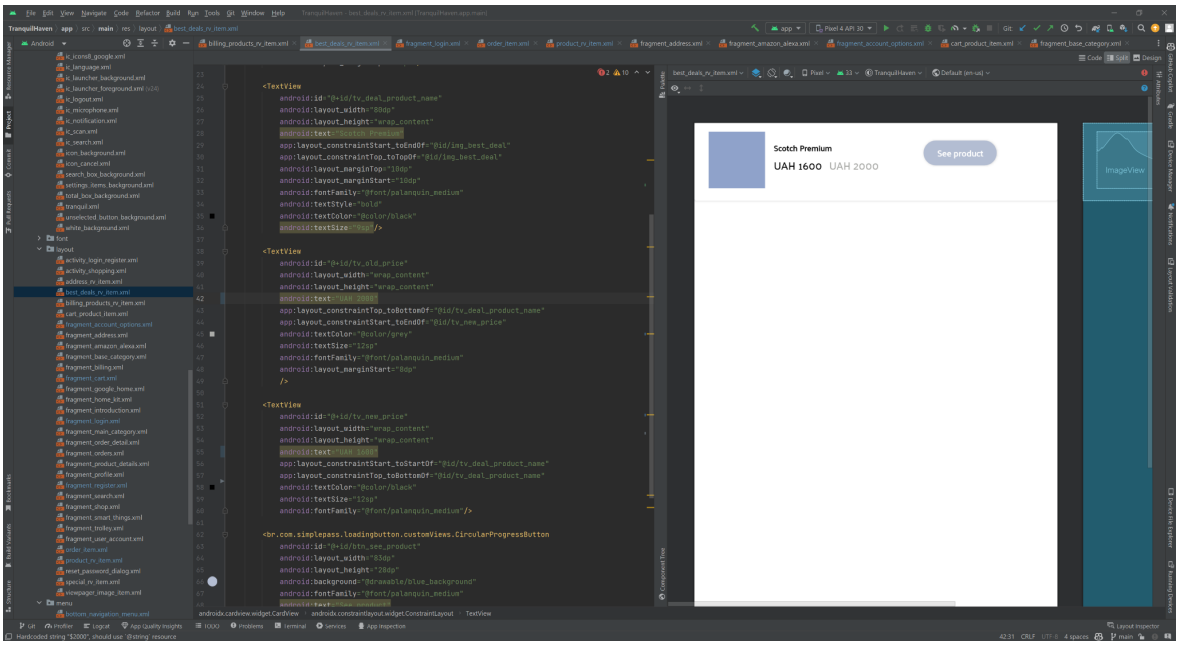


Рисунок 9 - Елемент списку який ми задаємо

На цьому прикладі є малюнок типу ImageView, 3 текста TextView (назва девайсу, ціна, та стара ціна якщо є знижка) та одна кнопка LoadingButtonAndroid.

Для відображення малюнків використана бібліотека Glide. Він допомагає отримати фотографії з серверу та кешувати в разі необхідності. Код щоб відобразити малюнок на місці ImageView виглядає так:

```
Glide.with(itemView)
  .load(product.images[0])
  .into(imgBestDeal)
```

2.3 Технології для навігації. Використані технічні засоби для роботи з екранами.

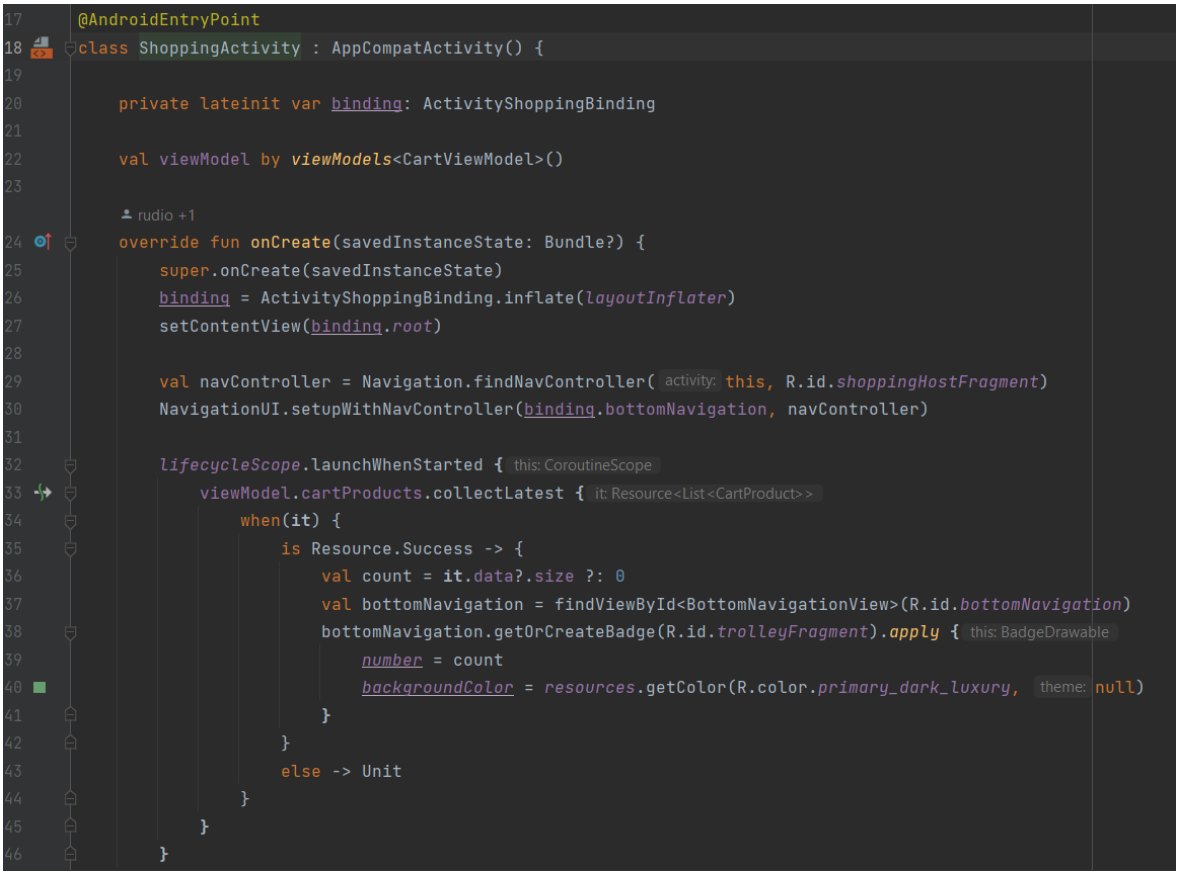
При створенні проекту мобільного додатку під операційну систему Андроїд, перше з чим розробник почне працювати - це головний екран. Звичайний головний екран - це **Activity** (надалі “Дії”). Головний екран магазину зображен на рисунку 10.

У розробці Android Activity є фундаментальним компонентом програми, яка забезпечує інтерфейс користувача для взаємодії з додатком. Це один із ключових будівельних блоків архітектури додатків Android, і він відіграє вирішальну роль у створенні бездоганної та інтуїтивно зрозумілої взаємодії з користувачем.

Дії можна використовувати для створення широкого діапазону компонентів інтерфейсу користувача, від простих екранів, що містять одне текстове поле, до складних екранів із кількома елементами та взаємодіями. Їх також можна використовувати в поєднанні з іншими компонентами Android, такими як служби та трансляційні приймачі, щоб створити більш повну роботу програми.

Однією з ключових переваг Activities є їх модульність. Їх можна використовувати для створення повторно використовуваних компонентів інтерфейсу користувача, які можна комбінувати з іншими діями для створення складних інтерфейсів користувача. Це полегшує створення складних і багатофункціональних програм, які легко підтримувати та оновлювати.

Приклад розробки функціоналу Activity зображено на рисунку 10 , де ми бачимо як прикріпляємо користувацький інтерфейс та інші компоненти для роботи з UI/UX



```
@AndroidEntryPoint
class ShoppingActivity : AppCompatActivity() {

    private lateinit var binding: ActivityShoppingBinding

    val viewModel by viewModels<CartViewModel>()

    rudio +1
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityShoppingBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val navController = Navigation.findNavController( activity: this, R.id.shoppingHostFragment)
        NavigationUI.setupWithNavController(binding.bottomNavigation, navController)

        lifecycleScope.launchWhenStarted { this: CoroutineScope
            viewModel.cartProducts.collectLatest { it: Resource<List<CartProduct>>
                when(it) {
                    is Resource.Success -> {
                        val count = it.data?.size ?: 0
                        val bottomNavigation = findViewById<BottomNavigationView>(R.id.bottomNavigation)
                        bottomNavigation.getOrCreateBadge(R.id.trolleyFragment).apply { this: BadgeDrawable
                            number = count
                            backgroundColor = resources.getColor(R.color.primary_dark_luxury, theme: null)
                        }
                    }
                    else -> Unit
                }
            }
        }
    }
```

Рисунок 10 - Головний екран Activity магазину

[8] Activity має чітко визначений життєвий цикл, який визначає, як вона поводиться на різних етапах свого існування. Життєвий цикл діяльності можна розділити на кілька етапів, а саме:

1. **Created**: коли дія створюється вперше, вона перебуває в стані створення. На цьому етапі користувач не бачить дії. На цьому етапі викликається метод onCreate() дії, і тут ініціалізуються змінні та інші ресурси.

2. **Started**: як тільки дія створюється, вона переходить у свій початковий стан. На цьому етапі активність видима користувачеві, але не має фокусу. На цьому етапі викликається метод onStart() дії, і тут дія готується стати видимою для користувача.

3. **Resumed**: коли активність перебуває у стані відновлення, вона видима користувачеві та має фокус. Це стадія, на якій активно виконується дія, і на неї можна відреагувати введенням користувача. У цей момент запускається метод onResume() активності, і саме тут активність вступає в контакт з користувачем.

4. **Paused**: коли діяльність втрачає фокус, але все ще є видимою для користувача, вона переходить у стан призупинення. На цьому етапі

діяльність більше не може відповідати на введення користувача. У цей момент викликається метод дії onPause(), і тут дія зберігає будь-які незбережені дані та готується до зупинки або відновлення.

5. **Stopped**: коли дія більше не відображається для користувача, вона переходить у стан зупинки. У цей момент викликається метод onStop() дії, і тут дія зберігає будь-які незбережені дані та готується до знищення або перезапуску.

6. **Destroyed**: коли дія більше не потрібна або користувач явно закриває її, вона переходить у знищений стан. На цьому етапі активність видаляється з пам'яті. У цей момент викликається метод onDestroy() активності, і тут звільняються будь-які ресурси, виділені діяльністю.

Хоч Activity і є екраном для того щоб показувати користувацький інтерфейс користувачу, але коли є великий проект, краще всього позбуватися часто використовувати саме цей компонент.

Для цієї задачі у нас є **Fragment**. Fragment — це основний компонент програми Android, який представляє частину інтерфейсу користувача з Activity. Його можна розглядати як модульний розділ діяльності, і його можна поєднувати з іншими фрагментами, щоб сформувати повний інтерфейс користувача.

Фрагменти пропонують кілька переваг перед Activities, що робить їх важливою частиною розробки Android. Однією з найважливіших переваг використання фрагментів є те, що їх можна повторно використовувати в різних видах діяльності. Це допомагає зменшити дублювання коду та зробити додаток більш модульним. Розбиваючи інтерфейс користувача на менші, більш зосереджені фрагменти, розробники можуть створити більш ефективну програму, яку можна підтримувати.

Ще одна перевага використання фрагментів полягає в тому, що їх можна використовувати для створення багатопанельного інтерфейсу користувача, який може бути особливо корисним на великих пристроях, таких як планшети. Поєднуючи кілька фрагментів в Activity, розробники можуть створювати вдосконалені інтерфейси користувача, які краще підходять для великих екранів. Це може допомогти забезпечити кращу взаємодію з користувачем і зробити додаток більш привабливим для користувачів.

Саме головне, що між фрагментами навігація набагато зручніше та ефективніше ніж між Activity. Для того щоб перейти на інший Activity без використання фрагментів, потрібно прописувати **Intent**.

[8] У розробці, Android Intent (наміри) - це об’єкт обміну повідомленнями, який використовується для зв’язку між компонентами програми, наприклад діяльністю, послугами та приймачами трансляції. Наміри можна використовувати, серед іншого, для запуску дій, запуску служб і надсилання трансляцій. Intent також можна використовувати для передачі даних між компонентами, що робить їх потужним інструментом для розробників Android.

Але переходячи на іншу активність, екран повністю знищується і створюється по нової коли користувач перейде назад. Якщо великий проект, то це дуже енергозатрата операція і не всі елементи ми зможемо перебрати на одному екрані. Для того щоб перейти на інший екран Activity, потрібно прописувати *(код створений лише для наглядності на мові Java)*:

Створюємо новий об'єкт Intent з поточного контексту (Activity) до цільового екрану (LoginRegisterActivity.class) та запускаємо нову активність, використовуючи створений Intent:

```
Intent intent = new Intent(LoginRegisterActivity.this, ShoppingActivity.class);
startActivity(intent);
```

У цьому прикладі CurrentActivity відповідає поточному екрану активності, з якого захочемо перейти, а TargetActivity - цільовому екрану, на який перейдемо. Розробник повинний замінити LoginRegisterActivityi ShoppingActivity на відповідні імена власного коду. При цьому важливо не забувати про те, щоб визначити ShoppingActivity та LoginRegisterActivity в файлі маніфесту проекту, додавши його в розділ <application>

У розробці Android файл AndroidManifest.xml є важливим файлом конфігурації, який описує основні функції програми та визначає кожен із її компонентів. Файл маніфесту має входити до кожного пакета програми, і він надає системі Android важливу інформацію про програму, таку як ім’я пакета програми, версія програми, дії та служби, які він містить, а також дозволи, які їй потрібні.

Файл маніфесту виконує кілька важливих завдань у розробці Android. По-перше, він надає системі Android спосіб ідентифікувати програму та керувати нею. Включаючи назву пакета та інформацію про

версію, файл маніфесту дозволяє системі однозначно ідентифікувати програму та переконатися, що її встановлено й оновлено належним чином.

По-друге, файл маніфесту визначає кожен із компонентів програми, наприклад дії, служби та отримувачі. Ці компоненти визначаються фільтром намірів, який визначає тип намірів, які вони можуть обробляти. Визначаючи компоненти у файлі маніфесту, система знає, які компоненти доступні, і може активувати їх за потреби.

Нарешті, файл маніфесту визначає дозволи, необхідні програмі для належного запуску. Дозволи використовуються для обмеження доступу програми до певних системних ресурсів, таких як камера чи Інтернет. Вказуючи дозволи, які вимагає програма, у файлі маніфесту, система може гарантувати, що програма має доступ, необхідний для належного функціонування.

Зрозумівши, що у великих проектах краще всього використовувати фрагменти, перейдемо до самої навігації між фрагментами. Навігація між фрагментами як і самі фрагменти знаходяться в бібліотеці Jetpack Navigation від Google, щоб її додати, потрібно прописати в dependency стрічки:

```
def nav_version = "2.5.2"
implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

В прикладі змінна nav_version це там де ми вписуємо версію бібліотеки для зручності, щоб не переносити значення на всі стрічки постійно коли бібліотека оновлюється.

Після додавання цієї бібліотеки можно створювати фрагменти та зв’язувати їх у вузли в графічному редакторі для навігації між ними.

Розглянемо як навігація працює в нашому додатку. В цьому пункті не буде розглянуто кожен екран і подивимось лише наші вузли у вигляді графу при використанні цих бібліотек

Всього у нас є 2 графа, граф авторизації та самого нашого магазину. Як і дизайн додатку, ми для цих вузлів використовуємо XML для побудови нашої навігації

Наш граф для навігації між екранами авторизації та реєстрації (рисунок 11):

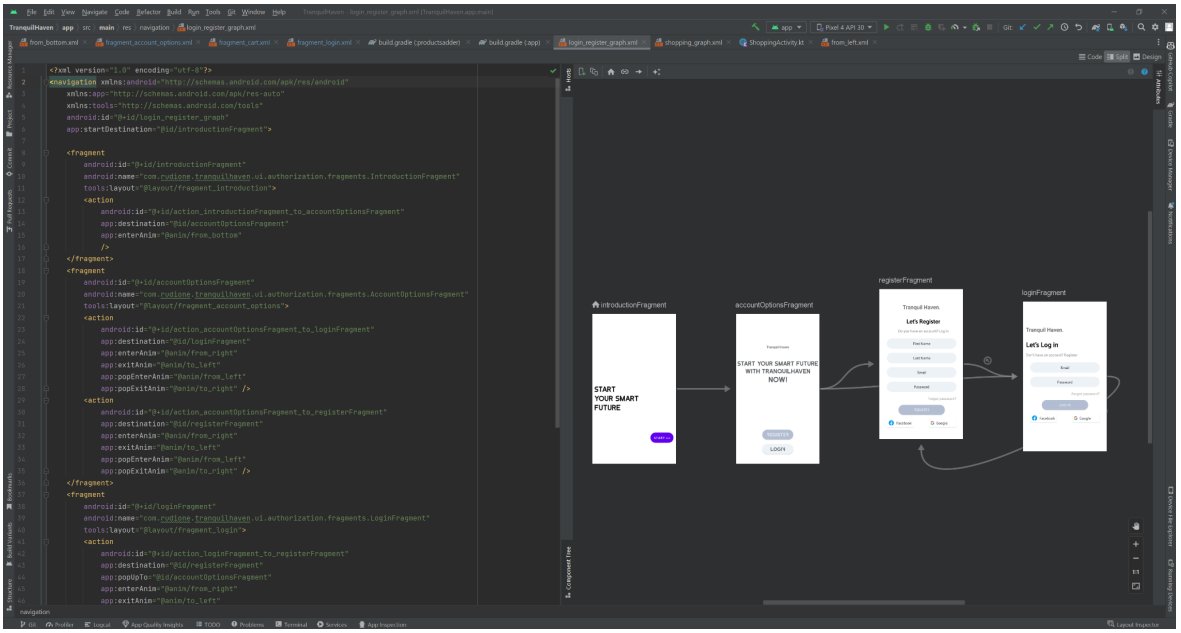


Рисунок 11 - Навігація між екранами авторизації

Наш граф для частини додатка з магазином, кошиком користувача, налаштуваннями користувача (рисунок 12:

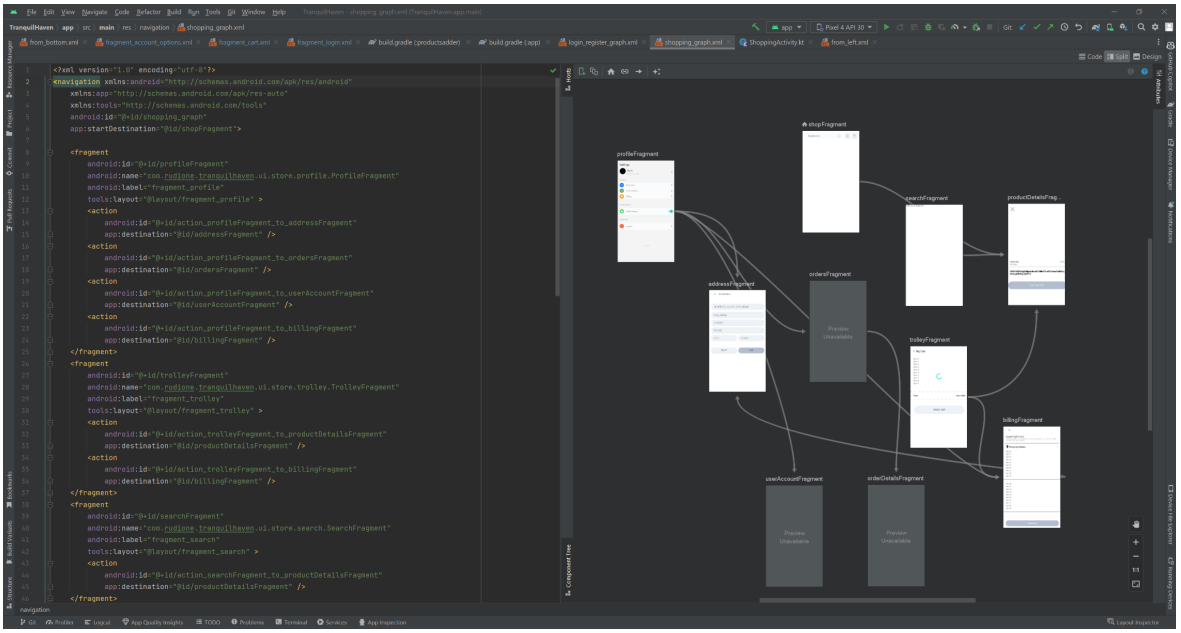


Рисунок 12- Навігація між екранами авторизації

2.4 Розвиток мов програмування під ОС Android

Обравши операційну систему під яку ми будемо розробляти мобільний додаток, а саме нею стала сама популярна операційна система Android, перейдемо до мов програмування.

Наразі є три офіційних мов програмування саме для нативної розробки, які підтримує та популяризує компанія Google: Java, Kotlin, C++.

Почнемо з першої. Мова програмування Java була створена в 1995 році компанією Sun Microsystems. Вона була розроблена для створення програмного забезпечення для різних платформ, включаючи комп'ютери, мобільні пристрої та інші електронні пристрої. Java була створена з метою забезпечення переносимості програм між різними платформами та забезпечення безпеки програм.

Java має багато переваг, таких як велика кількість бібліотек, що дозволяє розробникам створювати програми швидко та ефективно. Java також має велику спільноту розробників, що надає підтримку та допомогу у вирішенні проблем. Сама мова запозичує багато синтаксису з C та C++, але має простішу об'єктну модель і менше засобів низького рівня. У створенні мови Java було п’ять основних цілей:

1. Вона повинна використовувати методологію об'єктно орієнтованого програмування.
2. Вона повинна дозволяти виконувати одну і ту ж програму в декількох операційних системах.
3. Вона повинна містити вбудовану підтримку використання комп’ютерних мереж.
4. Вона повинна бути розроблена для безпечного виконання коду з віддалених джерел.
5. Вона повинна бути простим у використанні, вибираючи те, що вважалося хорошими частинами інших об’єктно-орієнтованих мов.

Переносимість є технічно важкою метою, і успіх Java у цій меті неоднозначний. Незважаючи на те, що дійсно можливо писати програми на Java, які ведуть себе узгоджено на кількох хост-платформах, велика кількість доступних платформ із незначними помилками чи невідповідностями спонукає деяких спародіювати гасло «напиши один раз, запусти будь-де». як «Напишіть один раз, наладьте всюди».

Другою мовою, яку ми розглянемо, є C++. Він використовується для розробки складних програм, бібліотек і коду низького рівня, коли нам потрібно більше працювати з самою операційною системою Android та її ядром. Однак розробка програм на C++ вимагає більше часу та зусиль через високий рівень складності мови.

Останнім і основним мовою, з яким ми будемо працювати в цій роботі, є Kotlin. Kotlin - це нова мова програмування для ОС Android. Це альтернатива мові Java, але має простіший синтаксис і більшу безпеку порівняно з Java. Створений з нуля для JVM (Віртуальна машина Java) та Android, він поєднує функції об’єктно-орієнтованого та функціонального програмування. Завдяки JVM код, написаний на Java, можна читати та працювати з ним за допомогою Kotlin, тому бібліотеки та технології, написані давно на Java, можна використовувати під час розробки в Kotlin.

У 2017 році на заході «Google IO» Google оголосив Kotlin офіційно підтримуваною мовою для Android. Відтоді величезна популярність Kotlin зростає з кожним днем.

Тому для кваліфікаційної роботи була обрана мова програмування Kotlin, з якою буде зручно реалізувати всі поставлені цілі та мати великі шанси розширити додаток у майбутньому без різноманітних проблем із сумісністю.

2.5 Технології проектування проекту. Архітектура проекту. Впровадження залежностей

При розробці проекту важливо пам’ятати про те, як правильно зв’язати весь код щоб він був читабельним та розширювальним. Для цього використовують шаблони проектування. В нашому проекті було використано саме MVVM (рисунок 13).

[11] MVVM (Model-View-ViewModel) - це архітектурний шаблон, який використовується в розробці програмного забезпечення, який відокремлює інтерфейс користувача (UI) програми від її бізнес-логіки.
У MVVM інтерфейс користувача розділений на три компоненти:

- Model: це дані та бізнес-логіка програми.
- View: це інтерфейс користувача програми.
- ViewModel: діє як міст між моделлю та представленням. Він надає представленню дані та бізнес-логіку моделі.

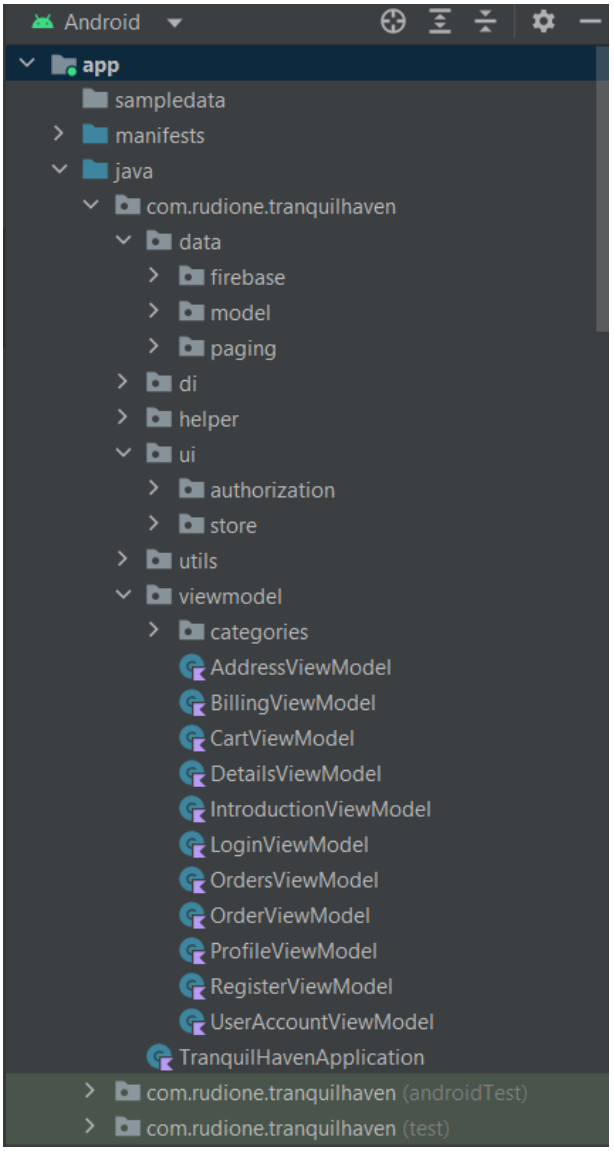


Рисунок 13 - Використання MVVM в цій роботі

В цьому проекті model - це папка data, де знаходиться вся бізнес-логіка розробки додатку, View - це папка з UI частиною, а ViewModel - папка в якій написані методи з якими ми працюємо щоб передати їх на екран.

Основна перевага використання MVVM полягає в тому, що це полегшує підтримку та тестування програм. Це означає окрему проблему між інтерфейсом користувача та бізнес-логікою, що робить код легшим для розуміння та зміни.

Особливу роль у цій архітектурі відіграє ViewModel. ViewModel — це компонент архітектури VVM, який відповідає за підтримку даних MV і стану елементів інтерфейсу користувача, коли діяльність знищується та

перетворюється. ViewModel зберігає дані в пам’яті, щоб їх можна було відновити після знищення та трансформації компонентів Android.

ViewModel дозволяє зменшити залежність між View та Model, завдяки чому код програми залишається чистим і забезпечує легку перевірку. ViewModel — незвичайний компонент для розробки Android-додатків на основі архітектури MVVM. Реалізацію ViewModel можно побачити в прикладі як ми створюємо авторизацію на рисунку 14.

```
@HiltViewModel
class LoginViewModel @Inject constructor(
    private val firebaseAuth: FirebaseAuth
) : ViewModel() {

    private val _login = MutableSharedFlow<Resource<FirebaseUser>>()
    val login = _login.asSharedFlow()

    private val _resetPassword = MutableSharedFlow<Resource<String>>()
    val resetPassword = _resetPassword.asSharedFlow()

    Rudione
    fun login(email: String, password: String) {
        viewModelScope.launch { this: CoroutineScope
            _login.emit(Resource.Loading())
        }
        firebaseAuth.signInWithEmailAndPassword(email, password)
            .addOnSuccessListener { it: AuthResult!
                viewModelScope.launch { this: CoroutineScope
                    it.user?.let { it: FirebaseUser
                        _login.emit(Resource.Success(it))
                    }
                }
            }.addOnFailureListener { it: Exception
                viewModelScope.launch { this: CoroutineScope
                    _login.emit(Resource.Failure(it.message.toString()))
                }
            }
    }
```

Рисунок 14 - Реалізація ViewModel на прикладі авторизації в додаток

Об’єкт ViewModel, який ми створюємо під час нашого Activity або його Fragment можуть спілкуватися за допомогою шаблону Observer (шаблон проектування звітів. Також відомий як «Спостерігач». Реалізований у механізмі класу, який дозволяє об’єкту цього класу отримувати повідомлення про зміни в стані інших об’єктів і таким чином спостерігати за ними) і отримати від класу ViewModel, що, відповідно до того, коли користувач знищує Fragment життєвого циклу циклу або Activity, наприклад, коли програма увімкнено, повертає значення або функцію, яку користувач мав до цього. Роботу з самим об’єктом ViewModel зображено на рисунку 15. На цьому малюнку

реалізован візуальний функціонал коли користувач авторизувався в наш додаток. При завантаженні екрану користувач бачить особливу анімацію, при помилці йому випливає помилка, та при успішної авторизації закінчується анімація завантаження, та перекидає на екран Activity з нашим головним екраном магазину.

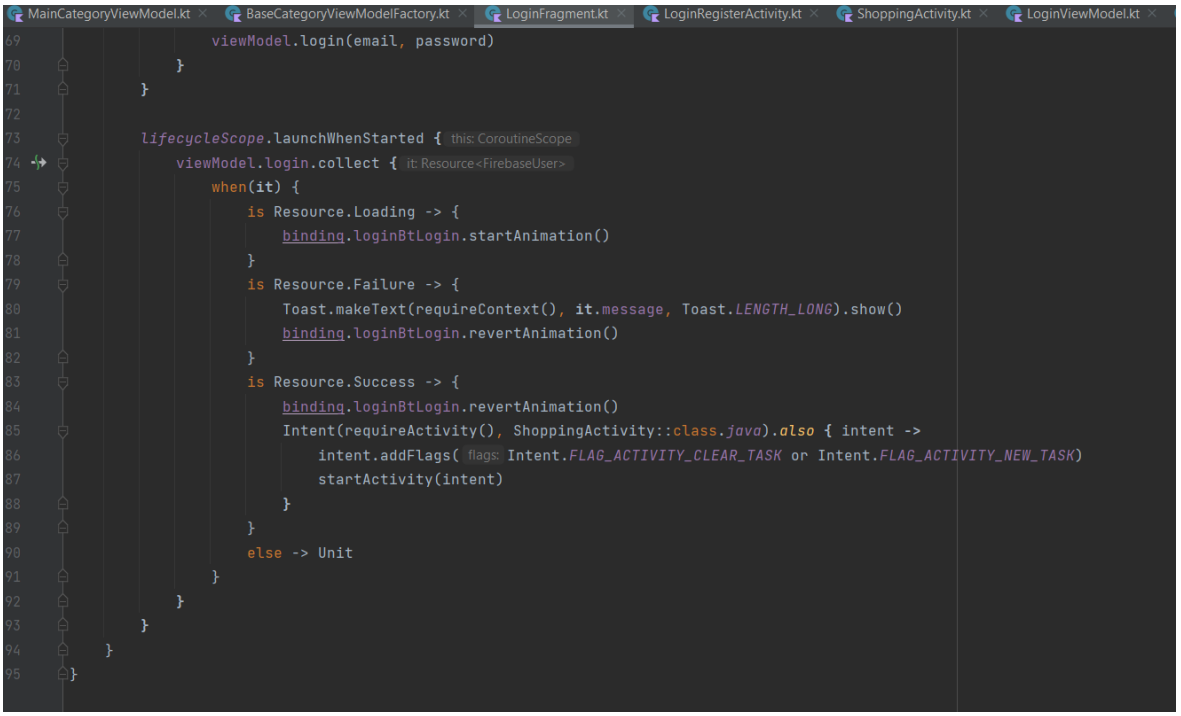
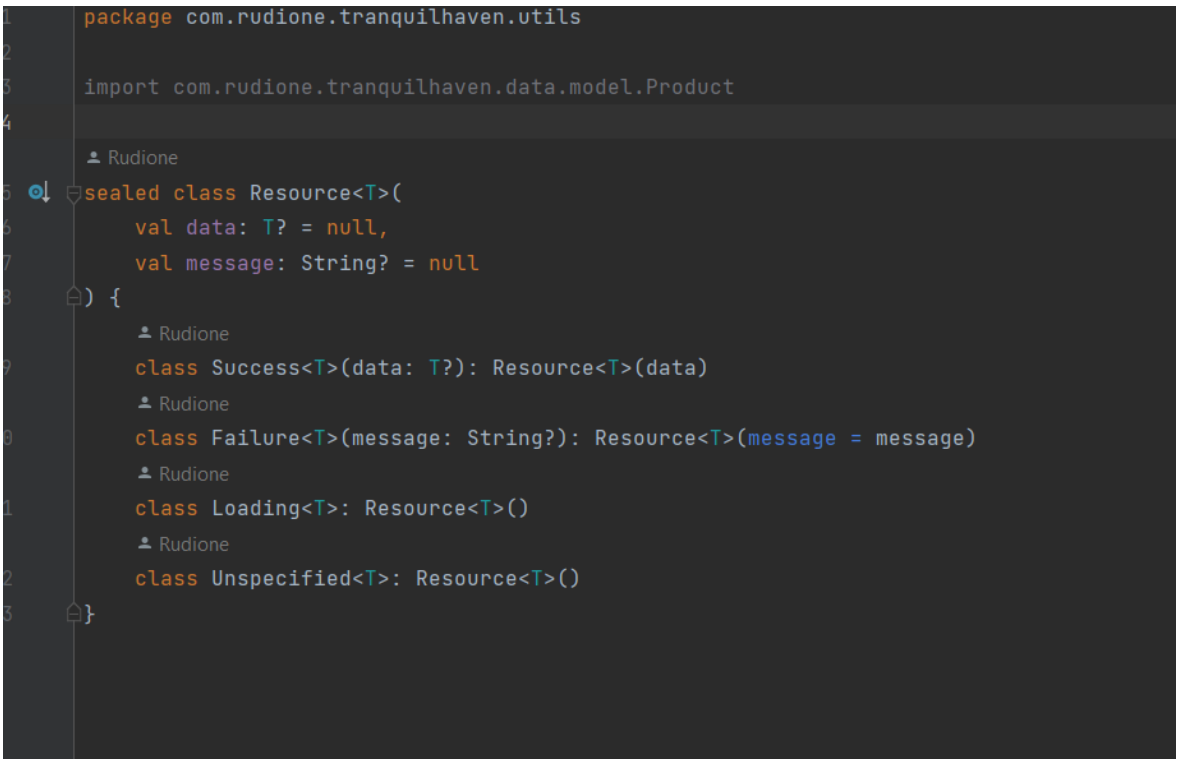


Рисунок 15 - Робота з об’єктом ViewModel та реалізація функціоналу

Щоб слідкувати за станом додатка, було створено особливий клас конструктор, де можна слідкувати в якому процесі працює додаток (рисунок 16). Для цього потрібно передавати у конструктор класу “Resource”, з яким працюємо, дані типу T (Дженерик, який вказує що типом може бути різні види типів). Є 4 типи стану:

- Success, де беремо дані та можемо їх передавати для подальшої роботи.
- Failure, де отримуємо повідомлення про помилку і можемо його показати користувачу.
- Loading, де працюємо з додатком під час як якийсь сеанс ще працює.
- Unspecified, коли ніякий сеанс не запущений.



```
package com.rudione.tranquilhaven.utils

import com.rudione.tranquilhaven.data.model.Product

Rudione
sealed class Resource<T>(
    val data: T? = null,
    val message: String? = null
) {
    Rudione
    class Success<T>(data: T?): Resource<T>(data)
    Rudione
    class Failure<T>(message: String?): Resource<T>(message = message)
    Rudione
    class Loading<T>: Resource<T>()
    Rudione
    class Unspecified<T>: Resource<T>()
}
```

Рисунок 16 - Особливий клас щоб дивитися за станом обробки даних

Для того щоб зручно було створювати та працювати з об’єктами, та впроваджувати елементи з “зовнішнього джерела”, було використано DI (Dependency Injection).

[12] Dependency Injection (Ін’єкція залежностей) — це шаблон проектування, який дозволяє створювати та використовувати об’єкти, не знаючи деталей того, як вони створені. У розробці Android впровадження залежностей використовується для керування залежностями між різними компонентами програми, такими як дії, фрагменти та сервіси (наприклад, надіслати повідомлення).

Hilt — це бібліотека для ін’єкції залежностей під Android. Вона забезпечує простий і легкий у використанні спосіб керування залежностями в додатку Android шляхом генерації коду для вас.

Використання Hilt у програмі Android допомагає зменшити шаблонний код і полегшує керування залежностями між різними компонентами.

Використовуючи ін’єкцію залежностей, можно зменшити зв’язок між різними компонентами програми, що спрощує зміну та розширення коду з часом.

Ось так виглядає клас Dependency Injection в проекті цієї дипломної роботи:

```
@Module
@InstallIn(SingletonComponent::class)
object AppModule {

  @Provides
  @Singleton
  fun provideFirebaseAuth() = FirebaseAuth.getInstance()

  @Provides
  @Singleton
  fun provideFirebaseFirestoreDatabase() = Firebase.firestore

  @Provides
  fun provideIntroductionSharedPreferences(
    application: Application
    ) = application.getSharedPreferences(Constants.INTRODUCTION_SP,
MODE_PRIVATE)

  @Provides
  @Singleton
  fun provideFirebaseCommon(
    firebaseAuth: FirebaseAuth,
    firestore: FirebaseFirestore
  ) = FirebaseCommon(firestore, firebaseAuth)

  @Provides
  @Singleton
  fun provideStorage() = FirebaseStorage.getInstance().reference }
```

- @Module: Анотація @Module використовується для позначення класу, який містить методи @Provides для створення та надання залежностей.
- @InstallIn(SingletonComponent::class): Анотація @InstallIn використовується для вказівки контексту, в якому будуть встановлені модулі. SingletonComponent::class вказує, що модуль буде встановлений в синглтон-компонент (шаблон проектування. Гарантує, що клас матиме тільки один екземпляр).

- @Provides: Анотація @Provides використовується для позначення методу, який постачає (створює або повертає) залежні об'єкти.
- @Singleton: Анотація @Singleton вказує, що надана залежність повинна бути створена та збережена як синглтон. Це означає, що весь час існування додатку буде використовуватися одна і та ж сама інстанція цієї залежності.

Написавши цей клас, можно зробити ін’єкцію (вбудувати об’єкт в клас) в конструкторі класу за допомогою анотації @Inject (рисунок 17)

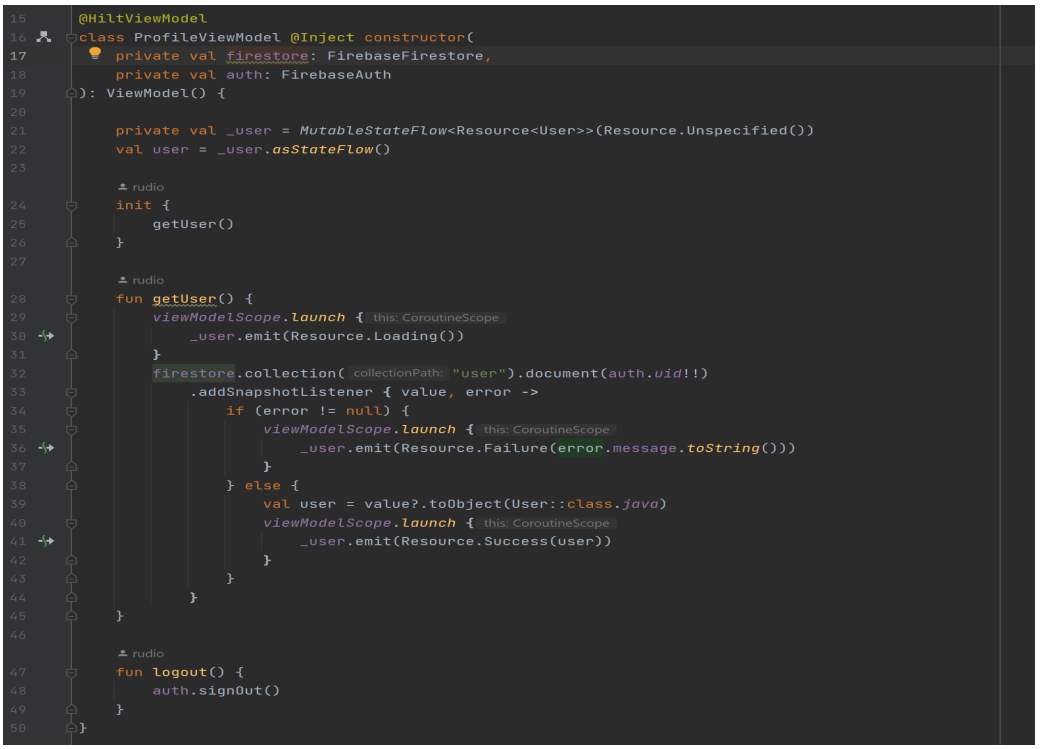


Рисунок 17 - Приклад впровадження залежності

Використовуючи Hilt, можно створювати в класі фрагменту чи Activity об’єкт ViewModel:

val viewModel by viewModels<ProfileViewModel>()

Коли користувач буде переходити по іншим екранам, новий об’єкт не буде створюватись з цим типом, що значно прискорює роботу з додатком.

2.6 Технології для роботи з Backend частиною

Проаналізувавши технології для роботи з Backend частиною (розробка бізнес-логіки продукту) під мобільні пристрої для операційної

системи Android, було одразу ж приділено увагу саме хмарній технології “Firebase”. Сама компанія Google рекомендує використовувати саме її, та користуватися бібліотеками цієї технології можно одразу ж як буде створено проект.

[13] Firebase – це платформа розробки мобільних і веб-додатків, розроблена Firebase, Inc. у 2011 році та пізніше придбана компанією Google у 2014 році. Firebase надає розробникам набір інструментів і послуг, які допомагають їм створювати високоякісні програми, розширювати базу користувачів і заробляти гроші. Firebase — це хмарна платформа, яка дозволяє розробникам зберігати та синхронізувати дані в реальному часі, автентифікувати користувачів і розміщувати їхні програми.

Firebase пропонує широкий спектр функцій, які роблять його популярним вибором для розробників. По-перше, Firebase пропонує Realtime Database, яка є хмарною базою даних NoSQL, яка зберігає та синхронізує дані в режимі реального часу. По-друге, автентифікація Firebase — це безпечна та проста у використанні система аутентифікації, яка дозволяє розробникам автентифікувати користувачів за допомогою електронної пошти та пароля, номерів телефонів і облікових записів у соціальних мережах. Хостинг Firebase швидкий, безпечний і простий у використанні, що робить його популярним вибором для розробників.

Розглянемо бібліотеки які було використано для бекенд частини за допомогою Firebase, але для початку додамо функцію дозволу інтернету щоб у користувача перевіряло наявність інтернету. Це важливо щоб можно було функціонувати між клієнт-серверною архітектурою. Добавимо для цього в AndroidManifest.xml, про який вже було сказано, одну стрічку:

<uses-permission android:name="android.permission.INTERNET" />

2.6.1 Аутентифікація користувача

Firebase Authentication є однією з найважливіших послуг Firebase, яка дозволяє забезпечити безпеку та конфіденційність користувачів програми. Firebase Authentication надає готові компоненти автентифікації користувачів, які можна легко інтегрувати в будь-яку платформу.

Однією з головних особливостей автентифікації Firebase є можливість автентифікації користувачів за допомогою різних методів, Аутентифікація Firebase дозволяє використовувати кілька методів аутентифікації в одній програмі, забезпечуючи більшу гнучкість і зручність для користувачів.

Друга особливість автентифікації Firebase — це можливість зберігати дані користувача та керувати ними в базі даних Firebase Realtime або Firestore. Він дозволяє зберігати профілі користувачів, їх налаштування та історію дій у програмі.

Щоб реалізувати аутентифікацію користувача, потрібно створити метод для роботи з цією бібліотекою. Також за допомогою Firebase Authentication можно реалізувати реєстрацію користувача (рисунок 18).

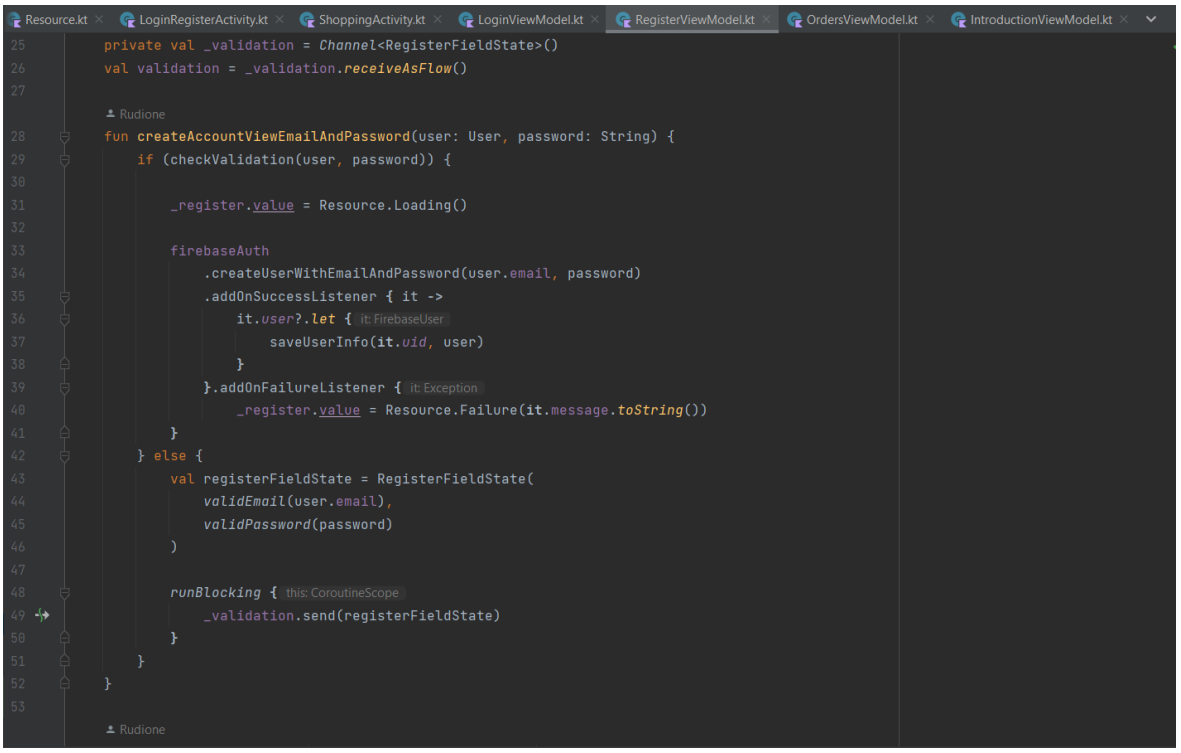


Рисунок 18 - Робота з Firebase Authentication для реєстрації користувача

Після чого потрібно зберегти дані користувача на сервері, для цього реалізований особливий метод, де ми передаємо на сервер всі важливі дані:

```
private fun saveUserInfo(userUid: String, user: User) {
  db.collection(Constants.USER_COLLECTION)
     .document(userUid)
```

```
        .set(user)
        .addOnSuccessListener {
            _register.value = Resource.Success(user)
        }.addOnFailureListener {
            _register.value = Resource.Failure(it.message.toString())
        }
    }
```

Нарешті, Firebase Authentication має вбудовані інструменти для захисту програм від атак на безпеку, таких як переповнення буфера та атаки SQL-ін'єкції. Аутентифікація Firebase дозволяє забезпечити безпеку даних і захистити їх від зловмисників.

Після успішної автентифікації користувача автентифікація Firebase базується на паролі. Токен містить інформацію про користувача та його права доступу до ресурсів програми. Коли користувач намагається отримати доступ до ресурсів програми, автентифікація Firebase перевіряє облікові дані та визначає, чи дозволено користувачеві доступ до цих ресурсів.

Аутентифікація Firebase також дозволяє встановлювати права доступу для різних груп користувачів, що дозволяє створити зручну та безпечну систему управління доступом до ресурсів програми.

Щоб зареєструвати користувачів у цьому сервісі Firebase, була створена модель, з якою ми будемо працювати. Ми називаємо цю модель «Користувач», і вона виглядає так:

```
 data class User(
   val firstName: String,
   val lastName: String,
   val email: String,
   val imagePath: String = ""
)
```

Користувач має задати ім'я, фамілію та пошту. З самого початку у користувача немає аватарки, замість неї буде стандарта фотографія, яку можно буде змінити в налаштуваннях.

Також можно переглядати зареєстрованих вже користувачів (рисунок 19), дивитися їх пошту, коли був створений обліковий запис та останній раз заходив у додаток. Як вже було сказано, у кожного користувача є свій ідентифікаційний номер.

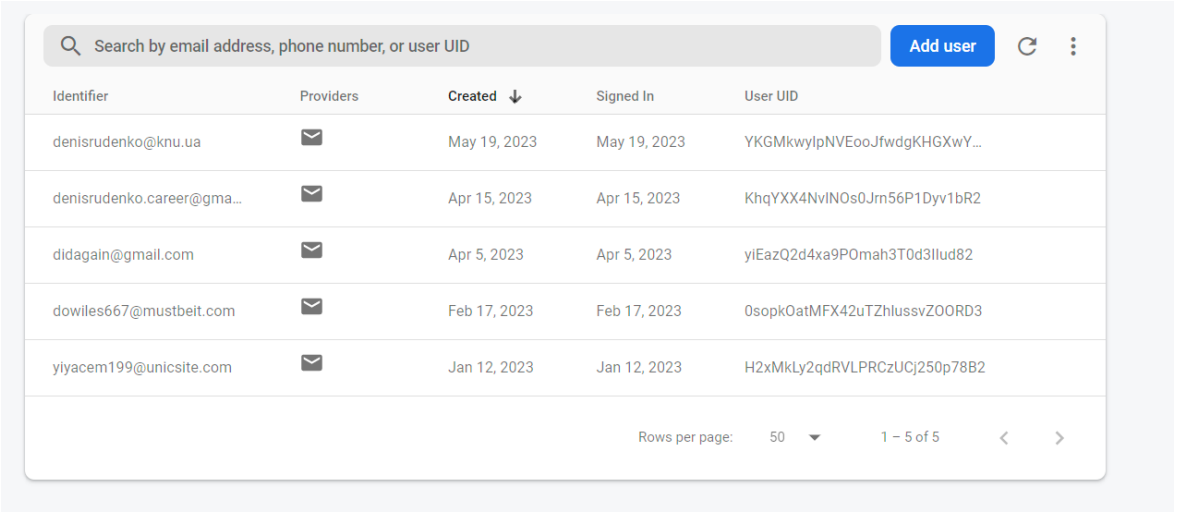


Рисунок 19 - список зареєстрованих користувачів

За допомогою цього сервісу, можно і задавати повідомлення для користувачів по різним функціям (рисунок 20), такі як: код-підтвердження на пошту, для того щоб зайти у додаток, відновити користувачу пароль тощо.

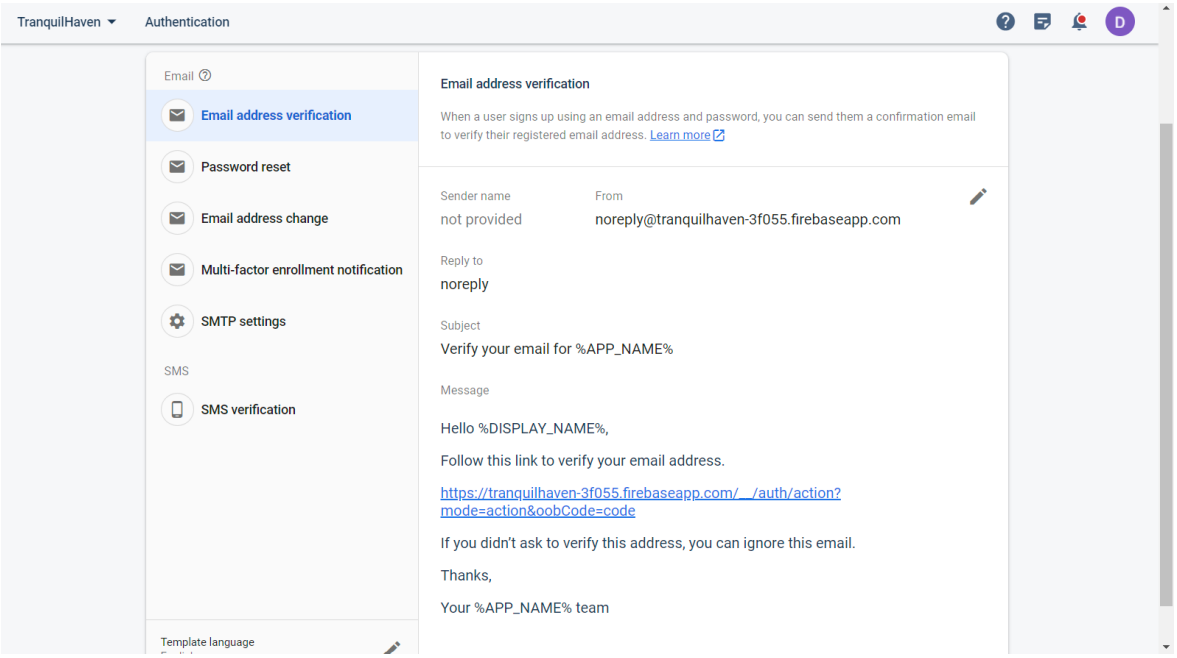


Рисунок 20 - розсилання повідомлення користувачу

2.6.2 Хмарне сховище

Для того щоб не навантажувати мобільний додаток чи пристрій користувача файлами, а включаючи те, що додаток оновлюється і товарів

все більшає, важливо під’єднати хмарне сховище, від якого буде зручно та швидко діставати дані щоб відобразити на екрані. Цю задачу ідеально робить Firebase Storage.

Firebase Storage — це сховище від Google, яке дозволяє зберігати та отримувати такі файли, як фотографії, відео та тексти, у хмарі. Firebase Storage забезпечує легкий доступ до файлів з будь-якого пристрою в Інтернеті. Firebase Storage дозволяє кожній програмі легко зберігати та отримувати файли з хмари безпосередньо з програми, що значно полегшує редагування.

Щоб отримувати файли з хмари з будь-якого пристрою, потрібно спочатку їх додати на сервер. Для цього був створений допоміжний додаток для персоналу, де особлива його функція - це створювати товари та зберігати фотографії у хмарі. Для цього був створений метод, де по вказаному шляху створюємо та зберігаємо дані:

```
private fun saveProduct() {
  val name = binding.edName.text.toString().trim()
  val category = binding.edCategory.text.toString().trim()
  val price = binding.edPrice.text.toString().trim()
  val offerPercentage = binding.offerPercentage.text.toString().trim()
  val productType = binding.edType.text.toString().trim()
  val description = binding.edDescription.text.toString().trim()
  val productDetails = binding.edDetails.text.toString().trim()
  val imagesByteArrays = getImageByteArrays()
  val images = mutableListOf<String>()

  lifecycleScope.launch(Dispatchers.IO) {
    withContext(Dispatchers.Main) {
      showLoading()
    }
    try {
      async {
        Log.d("test1", "test")
        imagesByteArrays.forEach {
          val id = UUID.randomUUID().toString()
          launch {
                                                  val imagesStorage =
productStorage.child("products/images/$id")
```

```
                val result = imagesStorage.putBytes(it).await()
                                                    val downloadUrl =
result.storage.downloadUrl.await().toString()
                images.add(downloadUrl)
            }
        }
    }.await()
} catch (e: java.lang.Exception) {
    e.printStackTrace()
    withContext(Dispatchers.Main) {
        hideLoading()
    }
}

val product = Product(
    UUID.randomUUID().toString(),
    name,
    category,
    productType,
    price.toFloat(),
    productDetails,
    if (offerPercentage.isEmpty()) null else offerPercentage.toFloat(),
    description.ifEmpty { null },
    if (selectedColors.isEmpty()) null else selectedColors,
    images
)
firestore.collection("Products").add(product).addOnSuccessListener {
    hideLoading()
}.addOnFailureListener {
    hideLoading()
Log.e("Error", it.message.toString()) } } }
```

Firebase Storage також надає захищені дані. Firebase Storage автоматично шифрує файли, щоб захистити дані. Firebase Storage також надає можливість блокувати доступ до файлів із хмари, допомагаючи зберегти дані користувачів у безпеці.

Вказавши шлях до папки в хмарному середовищі, ми можемо тримати свої файли в безпечному місці (рисунок 21). Також в Firebase можно слідкувати на головні сторінці скільки всього файлів було додано, коли, який розмір хмарного середовища становить на цей час та іншу всю активність.

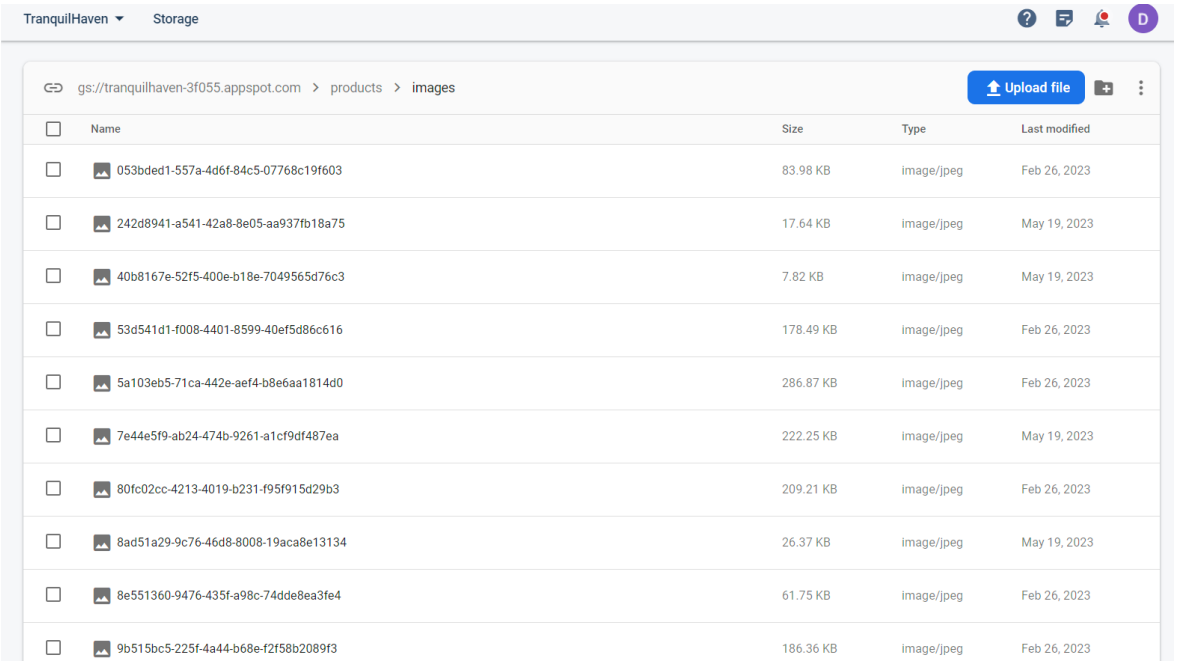


Рисунок 21 - хмарне середовище Firebase Storage

2.6.3 База даних у реальному часі

В ході розробки мобільного додатку для продажу електронних девайсів для розумних будівель, при аналізі було зрозуміло, що дуже важливо мати технологію, яка буде зчитувати дані закупівель, продукції та дані самих користувачів, передивлятись, редагувати та зберігати їх у реальному часі.

Для цього було використано технологію Firebase Firestore. Firebase Firestore — це база даних у реальному часі, яка є частиною платформи Firebase від Google. Firestore надає можливість зберігати та синхронізувати дані між різними пристроями та платформами. Особливості Firebase Firestore:

- Firestore забезпечує масштабований і гнучкий підхід до зберігання даних. Він дозволяє зберігати дані у вигляді колекцій і документів,

дозволяючи зручно структурувати дані та швидко виконувати пошук.

- Firestore працює в режимі реального часу. Це означає, що дані автоматично синхронізуються між пристроями та платформами. Якщо дані змінюються на одному пристрої, вони автоматично оновлюються на всіх інших підключених пристроях.
- Firestore підтримує автономний режим. Якщо пристрій втрачає підключення до Інтернету, Firestore може продовжувати працювати в автономному режимі.

Firestore підтримує аутентифікацію користувачів і забезпечує захист даних. Це означає, що ви можете контролювати доступ до даних і гарантувати захист приватної інформації, про саму аутентифікацію було сказано в іншому пункті. Як працює Firebase Firestore:

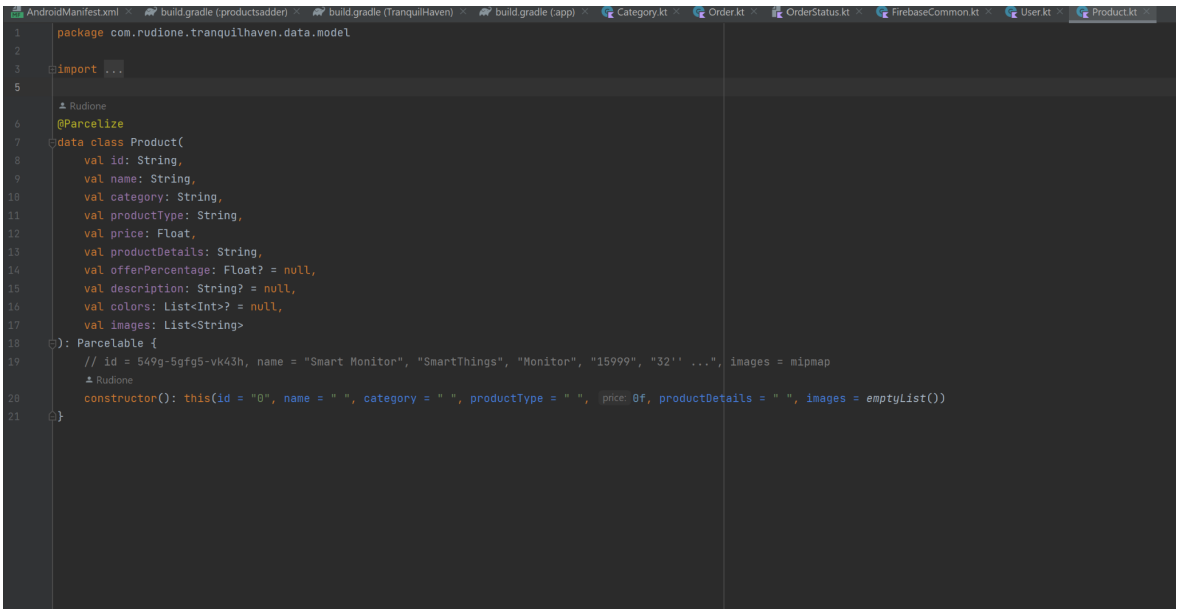
- Firestore працює за моделлю публікація-підписка. Коли дані на сервері змінюються, він автоматично сповіщає всі підключені пристрої, зареєстровані про зміни в цій області.
- Firestore використовує WebSocket для забезпечення швидкої та ефективної передачі даних між сервером і пристроями.
- Firestore має вбудований механізм кешування, що дозволяє зменшити кількість запитів до сервера та збільшити швидкість відповіді.

Для цього сервісу було створено методи, щоб зберігати дані order (купівлі), user (користувачі), products (продукції).

Для цього задано кожній моделі параметри. Так як модель user було вже розглянуто, роздивимось моделі даних products та order.

Модель даних products (рисунок 22) має такі дані, як:

- унікальний ідентифікаційний номер
- ім’я товару, та список малюнків
- категорія товару, а саме яка компанія: SmartThings, HomeKit та ін. (дуже важливий параметр, так як по ньому ми фільтруємо товари в магазині
- опис товару та тип продукції: телевізор, холодильник чи інше
- ціна, знижка на товар, та маленький опис
- колір (він працює лише як допоміжний засіб, самі товари на кольори не розділяються

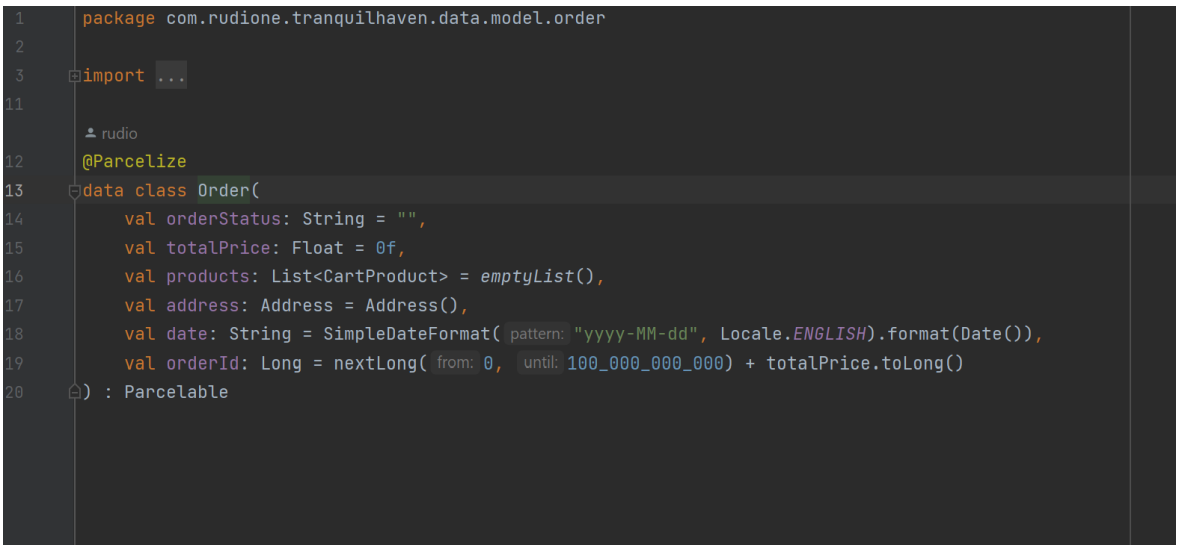


```
package com.rudione.tranquilhaven.data.model

import ...

Rudione
@Parcelize
data class Product(
    val id: String,
    val name: String,
    val category: String,
    val productType: String,
    val price: Float,
    val productDetails: String,
    val offerPercentage: Float? = null,
    val description: String? = null,
    val colors: List<Int>? = null,
    val images: List<String>
): Parcelable {
    // id = 549g-5gfg5-vk43h, name = "Smart Monitor", "SmartThings", "Monitor", "15999", "32'' ...", images = mipmap
    Rudione
    constructor(): this(id = "0", name = " ", category = " ", productType = " ", price: 0f, productDetails = " ", images = emptyList())
}
```

Рисунок 22 - модель даних product

Модель даних order (рисунок 23) має такі параметри:

- статус заявки
- кінечна ціна
- список замовлених товарів
- адрес користувача (яких також має параметр район, місто, вулиця)
- дата придбання, яка задається у форматі “2023.05.21”
- унікальний ідентифікатор покупки



```
package com.rudione.tranquilhaven.data.model.order

import ...

rudio
@Parcelize
data class Order(
    val orderStatus: String = "",
    val totalPrice: Float = 0f,
    val products: List<CartProduct> = emptyList(),
    val address: Address = Address(),
    val date: String = SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.ENGLISH).format(Date()),
    val orderId: Long = nextLong( from: 0, until: 100_000_000_000) + totalPrice.toLong()
) : Parcelable
```

Рисунок 23 - модель даних order

Всі ці моделі мають анотацію @Parcelize та повертають Parcelable. Це каже про те, що можно передавати ці моделі даних та самі дані між іншими екранами використовуючи Jetpack Navigation для навігації.

Щоб отримати товари з нашої бази даних у реальному часі, створено методи, де вказавши шлях до необхідних товарів, отримуємо в список товарів магазину необхідні девайси з усіма їх даними типу Product (рисунок 24).

```
73
    Rudione
74  fun fetchBestProducts() {
75      if (!pagingInfo.isPagingEnd) {
76          viewModelScope.launch { this: CoroutineScope
77              _bestProducts.emit(Resource.Loading())
78          }
79          firestore.collection( collectionPath: "Products").limit(pagingInfo.page * 10).get()
80              .addOnSuccessListener { it: QuerySnapshot!
81                  val bestProducts = it.toObjects(Product::class.java)
82                  pagingInfo.isPagingEnd = bestProducts == pagingInfo.oldBestProducts
83                  pagingInfo.oldBestProducts = bestProducts
84                  viewModelScope.launch { this: CoroutineScope
85                      _bestProducts.emit(Resource.Success(bestProducts))
86                  }
87                  pagingInfo.page++
88              }.addOnFailureListener { it: Exception
89                  viewModelScope.launch { this: CoroutineScope
90                      _bestProducts.emit(Resource.Failure(it.message.toString()))
91                  }
92              }
93      }
94  }
95
```

Рисунок 24 - Отримання даних з бази даних у реальному часі

Для того щоб відображати та обробляти дані зручно, та не заважати процесу відображення даних на екрані потрібно розбивати ці задачі на потоки.

В сучасному програмуванні асинхронні операції стають все більш популярними. Щоб забезпечити ефективну роботу з даними, одним із способів є використання Kotlin Coroutines та Coroutines / Flow.

Coroutines (надалі “співпрограми”) - це особлива технологія для виконання фунції асихронно, яку можна відкладати та відновлювати під час виконання, оскільки кожен потік виконання потребує власних ресурсів.

Крім того, в Kotlin з'явилася нова бібліотека - Flow. Він дозволяє зручно і ефективно працювати з потоками даних і вирішити проблему зворотного тиску. Зворотний тиск - це проблема, коли виробник створює дані швидше, ніж споживач може їх обробити. Щоб запустити Coroutines Flow, потрібно визвати метод та додати до нього метод “launch”, що запустить його за допомогою цієї технології. Приклад цього є додавання нової продукції в магазин з серверу. Щоб прискорити роботу

підгруження файлів з серверу, коли працює робота підгруження користувацького інтерфейсу, був створений метод:

```
private fun addNewProduct(cartProduct: CartProduct) {
firebaseCommon.addProductToCart(cartProduct){cartProduct,
exception ->
  if (exception == null) {
viewModelScope.launch{
_addToCart.emit(Resource.Success(cartProduct!!)) }
  } else {
viewModelScope.launch{
_addToCart.emit(Resource.Failure(exception.message.toString())) }
  }
}
}
```

Інші методи різного функціоналу будуються також через корутини, щоб прискорити роботу з додатком

Таким чином, співпрограми та Flow є потужними та зручними інструментами, які дозволяють ефективно працювати з асинхронними потоками даних.

РОЗДІЛ 3. РОБОТА З СИСТЕМОЮ

В результаті виконання бакалаврської роботи було створено 15 екранів. Вони були розділені на 2 версії:

1. Версія для користувачів, яка була розділена на 2 частини навігацій:
   - Навігація між екранами авторизації та реєстрації, яка складається з однієї Activity, який містить 4 фрагмента (екрани які бачить користувач і може на них перейти).

   - Навігація між екранами самого додатку, який містить магазин, кошик користувача та його профіль. В сумі він має 1 Activity та 10 фрагментів.

2. Версія для персоналу, яка містить 1 екран Activity

3.1 **Додаток для користувачів**

Навігація між екранами користувачів, як вже було сказано, складається з 2 графів, в яких ми задаємо і показуємо як функціонують наші екрани. Для переходу між екранами нам потрібно при натисканні потрібної кнопки написати:
findNavController().navigate(R.id.*action_profileFragment_to_ordersFragment*),
де ми вказуємо з якого екрану переходимо на який (в нашому разу з екрану профілю переходимо до списку всіх замовлень.

Для екранів авторизації було створено навігаційний граф у файлі **login_register_graph.xml**, а для графу самого магазину **shopping_graph.xml**.

3.1.1 **Екрани авторизації та реєстрації в додатку**

Запустивши вперше додаток, користувач отримує Introduction Fragment, який вітає користувача, в ньому є 3 TextView (3 текстових поля) та одну кнопку щоб перейти далі (рисунок 25). Натиснувши далі,

користувача перекине на екран Account Option Fragment, де йому буде надасть вибір між тим щоб авторизуватися чи зареєструватись у додатку, якщо він ще не має свого облікового запису (рисунок 26). Він має 4 TextView та 2 кнопки.

рисунок 25 - Introduction Fragment

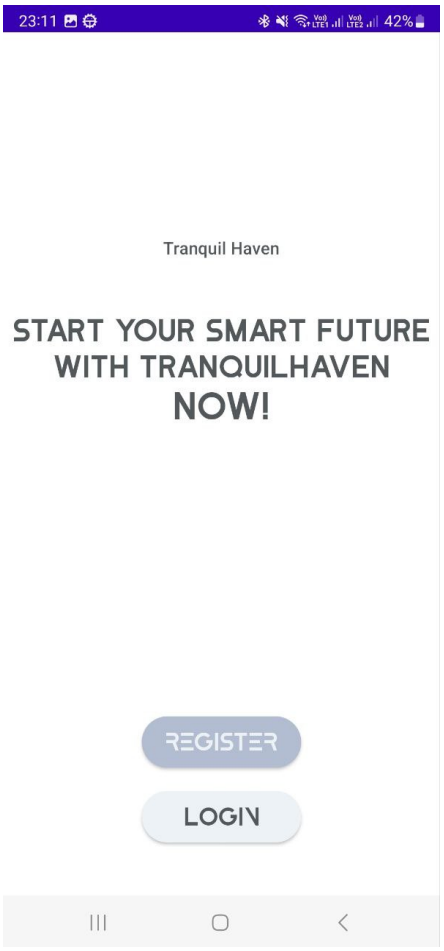


рисунок 26 - Account Options Frag.

Обравши варіант щоб авторизуватися у додатку, користувача перекине на екран (рисунок 27) з 4 текстовими полями, 1 кнопкою, та 2 полями для вводу, де користувачу знадобиться увійти у свій обліковий запис використовуючи свої дані у вигляді емейлу та паролю.

Якщо користувач не має свого облікового запису та вирішив зареєструватися, то він потрапить на екран реєстрації (рисунок 28), який складається також з 4 текстовими полями, 1 кнопкою але вже з 4 полями для вводу, де потрібно буде вести його персональне ім’я, фамілію, email та password.

Важливо помітити, що якщо користувач натисне на “Do you have account? Log in” чи “Don’t have an account? Register”, - його перекине на протилежний екран для того, щоб зберегти час.

В додатку також було додано валідацію паролю та пошти, якщо користувач щось з цього неправильно вів, йому покаже помилку та не дасть ввійти чи зареєструватися в магазин (рисунок 29).

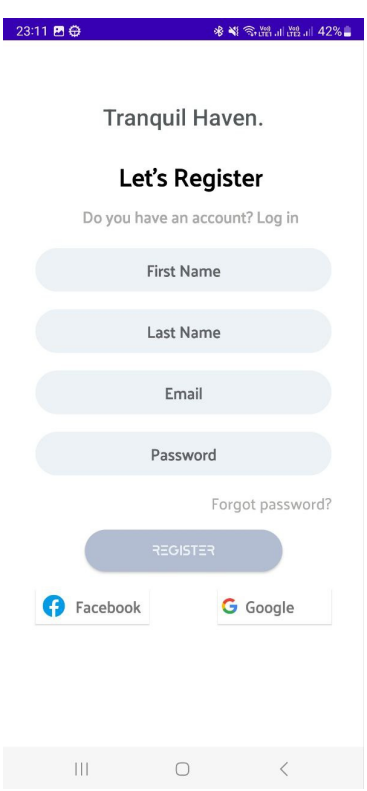
Трапляється таке, що користувач не пам’ятає свій пароль, та не хочеться втрачати свій обліковий запис. Для цього було створено функцію, щоб користувач міг відновити свій власний пароль ввівши свою пошту та перейшовши за новим посиланням який прийде на пошту.

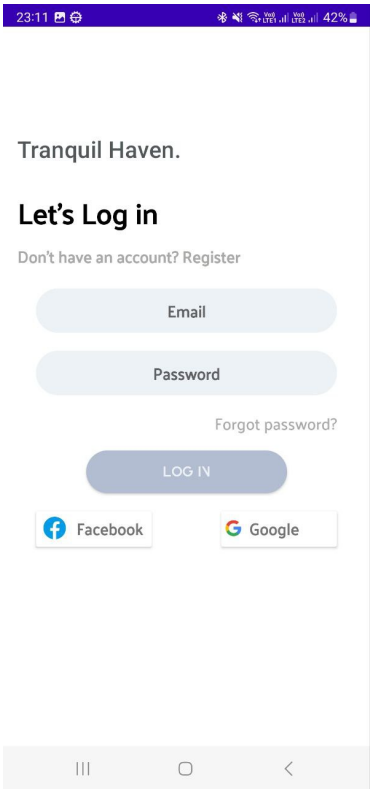
Саме вікно з відновленням паролю виглядає як спливаюче віконце, яке відкривається на екрані де знаходиться користувач (рисунок 30).

Якщо ви все зробили правильно, у вас вдалося авторизуватися чи зареєструватися, то вітаємо, вас перекине на новий граф де знаходиться навігація самого магазину і ви побачите сам екран магазину.

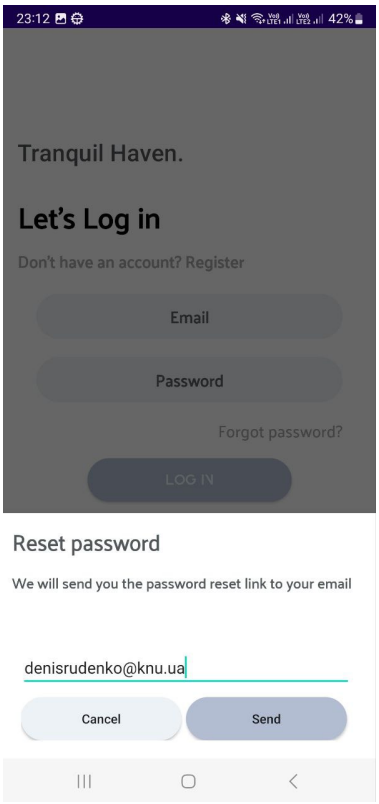
Розділення на графи було зроблено для того, щоб користувач якщо вже один раз авторизувався, то йому не потрібно було цього робити в інший раз коли захоче після закриття додатку відкрити його знову.

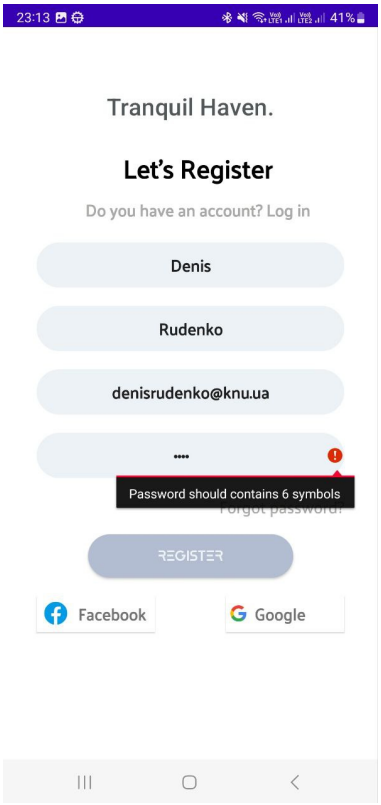
Після цього користувач може спокійно користуватися додатком, а якщо забажає вийти з нього, то має таку функцію в своєму профілі.





рисунки [27-28] - екрани Register Fragment та Login Fragment





рисунки [29-30] - спливаюче вікно з відновленням паролю та валідацією введених даних

3.1.2 Екрани магазину, кошику та профілю користувача

Авторизувавшись в свій обліковий запис, чи зайшовши у додаток зі свого аккаунту, користувач бачe основний екран магазину Shop Fragment (рисунок 31).

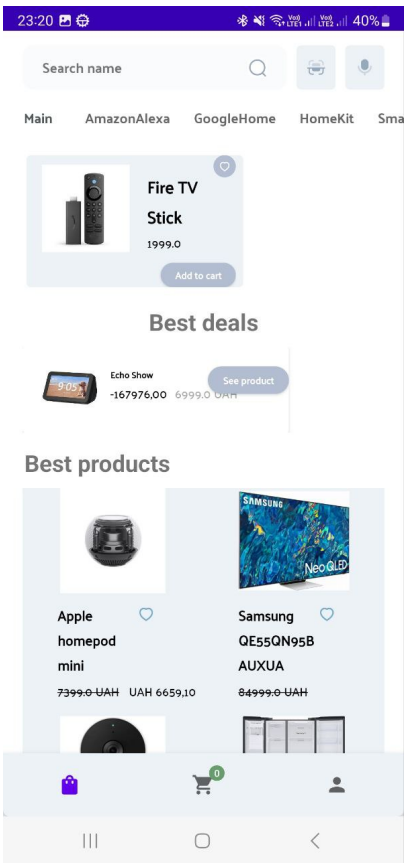


Рисунок 31 - екран Shop Fragment

В ньому знаходиться поля вводу тексту, сканування товарів по коду, голосовий асистент вибір між компаніями офіційні пристрої яких ми хочемо використовувати, та 3 списку товарів. Важливо! В даній версії додатку пошук по всім товарам, голосовий асистент та сканування товарів не передбачено, але розширюючи функціонал і оновлюючи додаток ці функції будуть добавлені першою чергою.

В магазині є 3 списку товарів, які передбачають:

1. Special Products, спеціальні продукти які стоять в самому горі, це можуть бути новинки або самі популярні товари між юзерами
2. Best Deals, товари які мають наразі самі вигідні по ціні, або йде розпродаж на них
3. Список всіх товарів

Натиснувши на товар, ми можемо перейти на екран Product Details Fragment (рисунок 32) з детальним описом, малюнками, які можемо гортати. Сам екран містить ViewPager, з слайдером щоб гортати малюнки; 2 кнопки, для додавання товару у кошик та кнопкою щоб перейти на попередній екран; 4 TextView з ціною, опису, на назвами товару.

Натиснувши кнопку “Add to cart”, ми можемо додати товар до кошику користувача і нам напише Toast (спливаюче повідомлення знизу), що ми додали товар у кошик (рисунок 33).

Рисунок 32 - Product Details Fragment

Рисунок 33 - Toast при додані до кош.

Перейшовши до екрану кошику (рисунок 34), користувач може побачити всі товари які він додав наразі до кошику, та суму замовлення у гривнях. На екрані знаходиться 2 кнопки, для переходу на попередній екран та замовлення товару; Список товарів, та їх кількість у вигляді TextView. Користувач може змінювати кількість товарів прямо на цьому екрані; 1 ImageView, який показує малюнок товарів; та текст на якому написана сума замовлення у чеку.

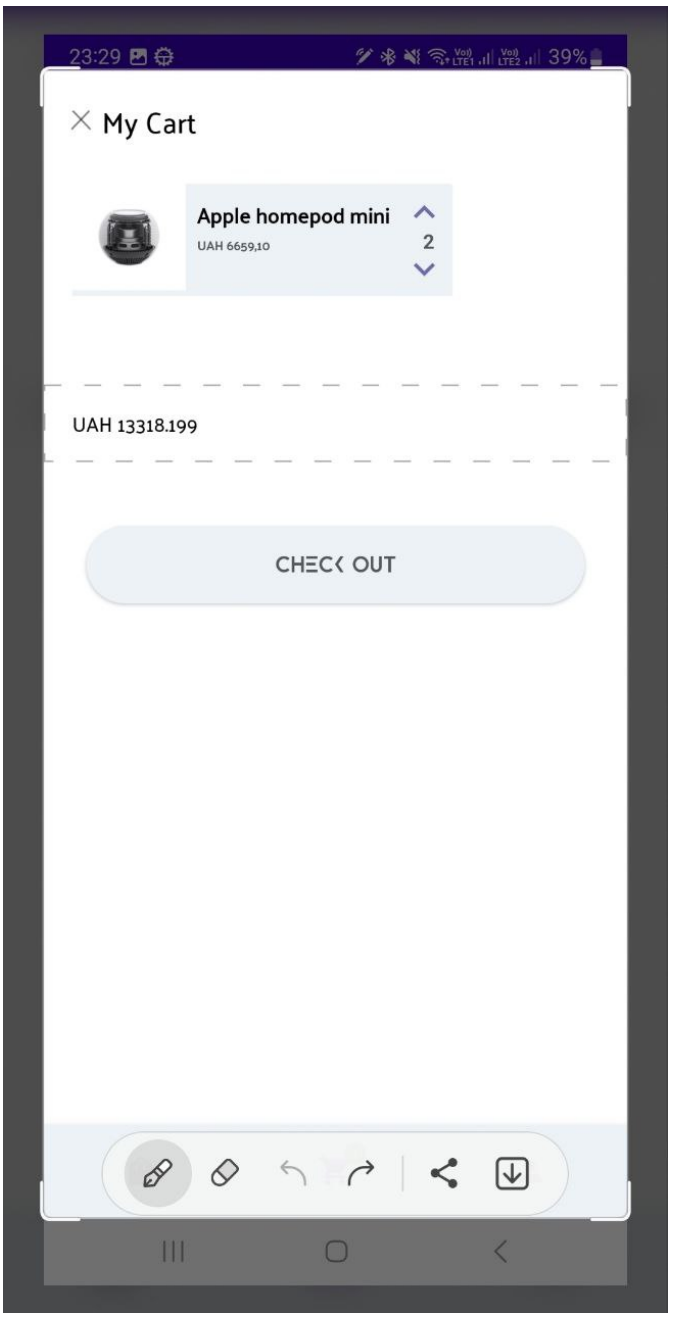


Рисунок 34 - Cart Fragment, кошик користувача

Після того, як користувач натисне кнопку “Check out”, якщо в його профілі не буде добавлено жодного адресу куди доставляти товар, йому запропонують добавити новий адрес (рисунок 35), перекинув до екрану Address Fragment, це важлива операція, без якої користувачу не дасть прибрати товар. На цьому екрані юзеру потрібно ввести дані для 6 полів, після чого, якщо користувач додав адресу, його перекине вже на екран з замовленням товару Billing Fragment (рисунок 36). На цьому екрані йому знадобиться обрати адресу, одну з яких він додав до себе в обліковий запис.

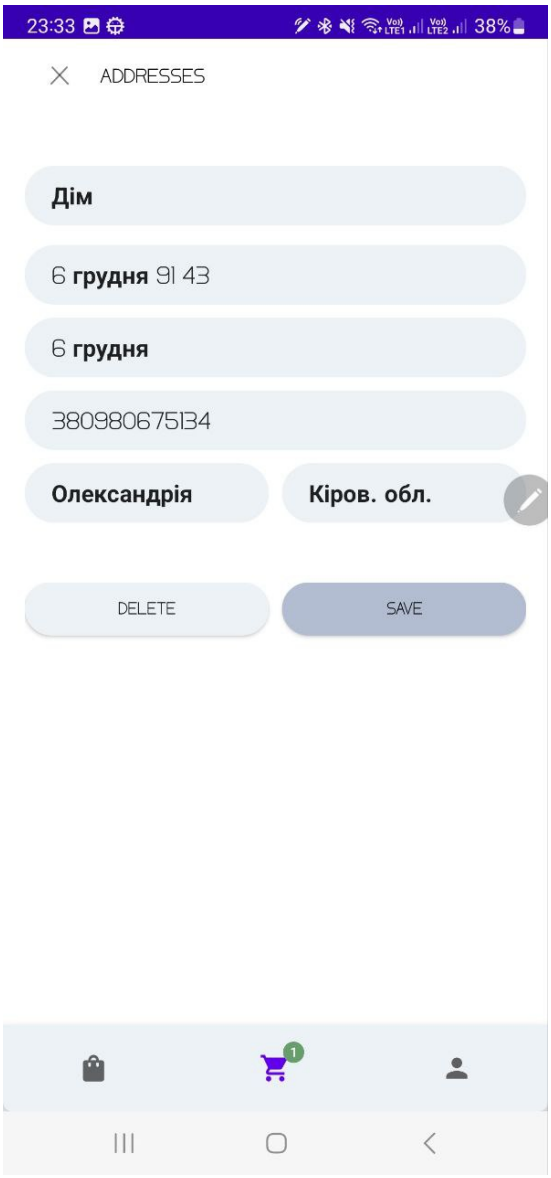


Рисунок 35 - Address Fragment

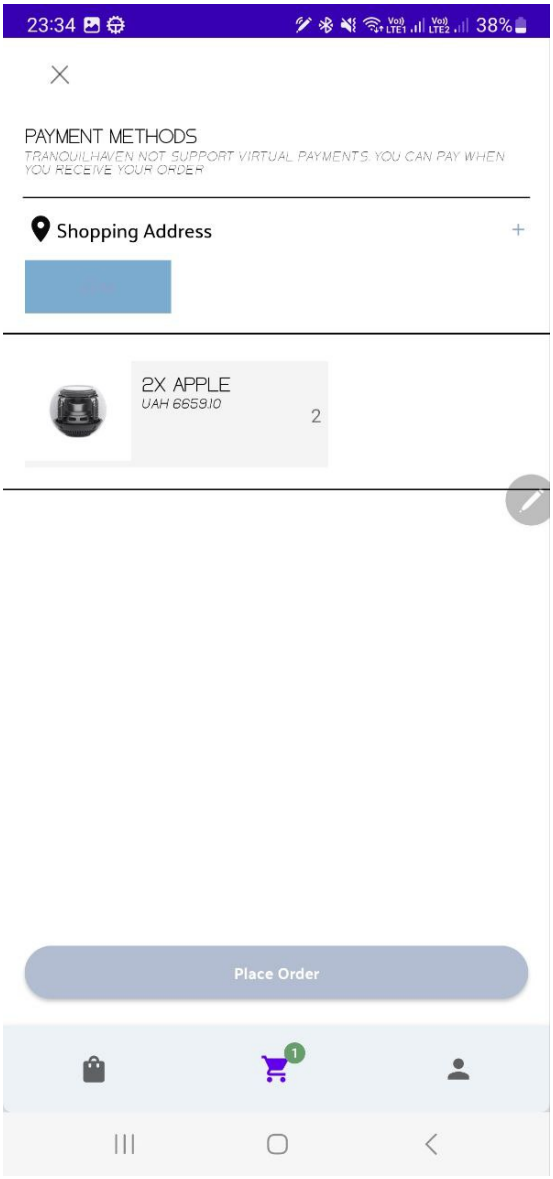


Рисунок 36 - Billing Fragment

Після того, як користувач натиснув кнопку “Place Order”, йому на аккаунт добавиться в історію, використовуючи технологію Firebase Firestore (рисунок 37), дане замовлення, за яким він може переглядати у будь який час перейшовши до свого профілю у мобільному додатку. Таким самим чином додається і адреса користувача, якщо перейти до його його облікового запису у базі даних

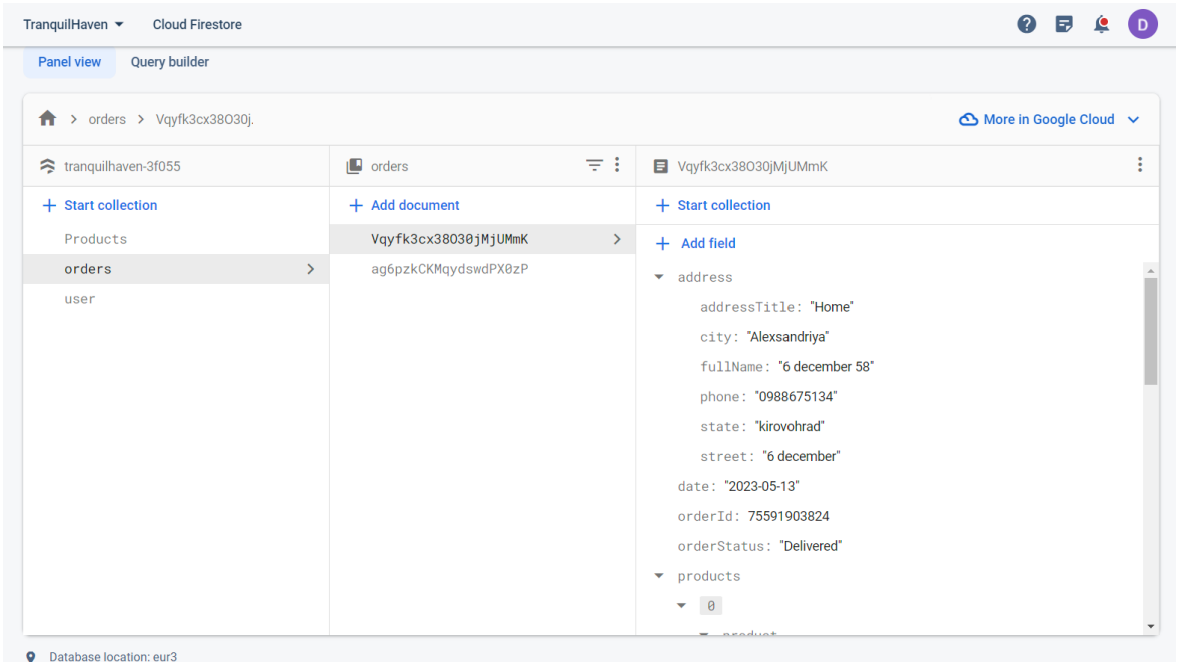


Рисунок 37 - База даних де зберігаються замовлення користувачів

Останній екран з нижньої панелі - це екран профілю користувача Profile Fragment (рисунок 38). На цьому екрані зображена: фотографія користувача, навіть якщо її немає то показується картина “Malevich Square”; дані користувача, такі як його персональне ім’я та фамілія. Користувач в будь який час може змінити свої персональні дані, натиснувши на своє ім’я чи на кнопку справа від тексту “edit personal details”.

З екрану профілю користувача, він може перейти до екрану всіх замовлень (рисунок 39), екран відстеження замовлень (рисунок 40) та екран з адресами. Користувач може добавити адрес стільки, скільки побажає.

Також на екрані користувач може керувати повідомленнями та вийти з свого особистого аккаунту.

Внизу написана версія додатку (наразі версія магазину “1.0”)

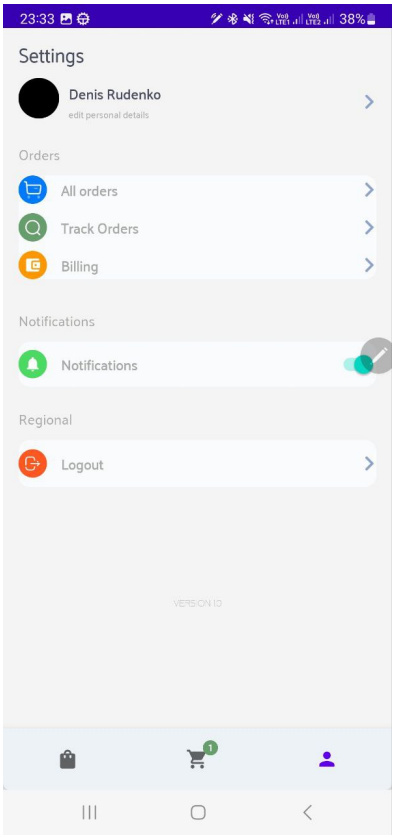


Рисунок 38 - Profile Fragment, обліковий запис користувача

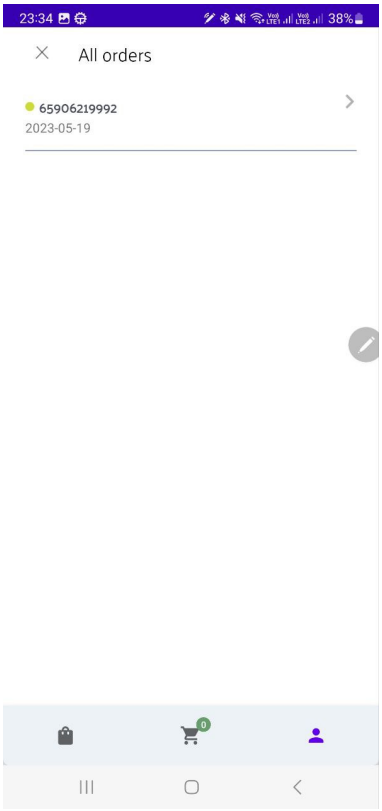


Рисунок 39 - All Orders Fragment

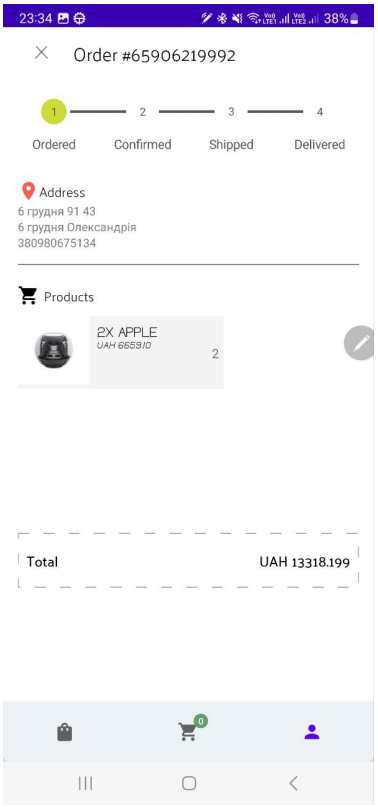


Рисунок 40 - Order Fragment

Важливий елемент розробки мобільного додатку, це тестування. Саме головне при цьому, перевірити, що екран коректно відображає елементи на екранів. Ми можемо переконатись, що завдяки правильному підходу розробки мобільного додатку, сам UI не зламається якщо користувач поверне свій екран (рисунок 41). Цим підходом можно зрозуміти, що елементи на екрані також будуть коректно відображатись на планшетах та інших девайсах, що мають операційну систему Android.

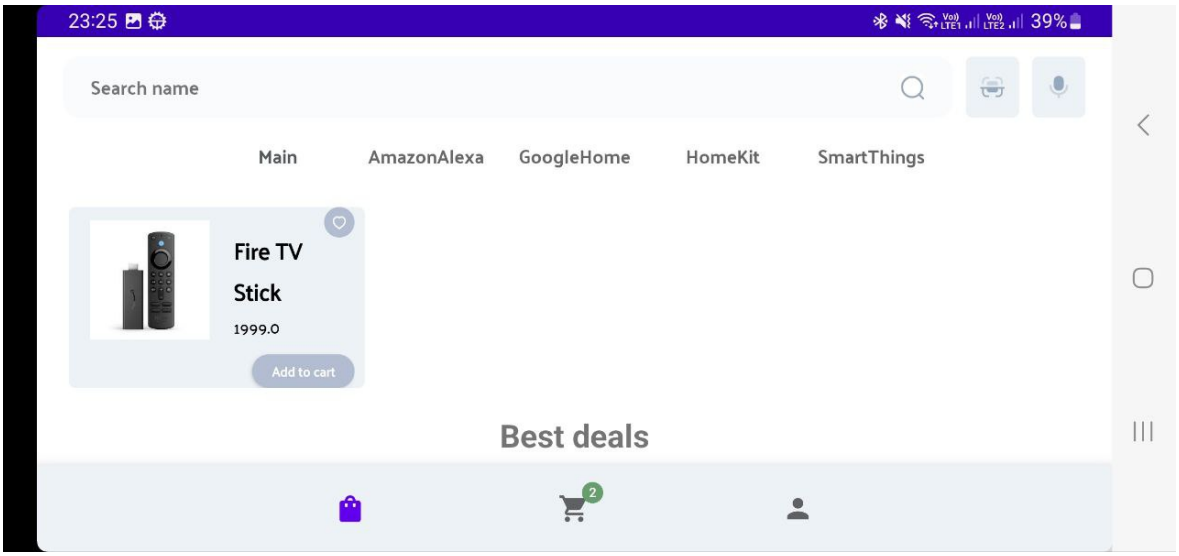


Рисунок 41 - тестування мобільного застосунку на поворот екрану

3.2 Додаток для персоналу

Не дивлячись на те, що мобільний додаток для персоналу складається всього лиш з 1 екрану MainActivity (рисунок 42), написаної логіки та функціоналу в цій версії достатньо

Ця версія додатку позбавляє нас проблеми як додавати товар безпечно і щоб все працювало ефективно та швидко.

Всі добавлені елементи додаються в режимі реального часу, де за допомогою Firebase Firestore додаємо товар в магазин та зберігаємо файли за допомогою Firebase Storage.

Для цього потрібно ввести всі необхідні дані в полях вводу (рисунок 43), щоб зберегти модель product, який вже було розглянуто в минулих етапах.

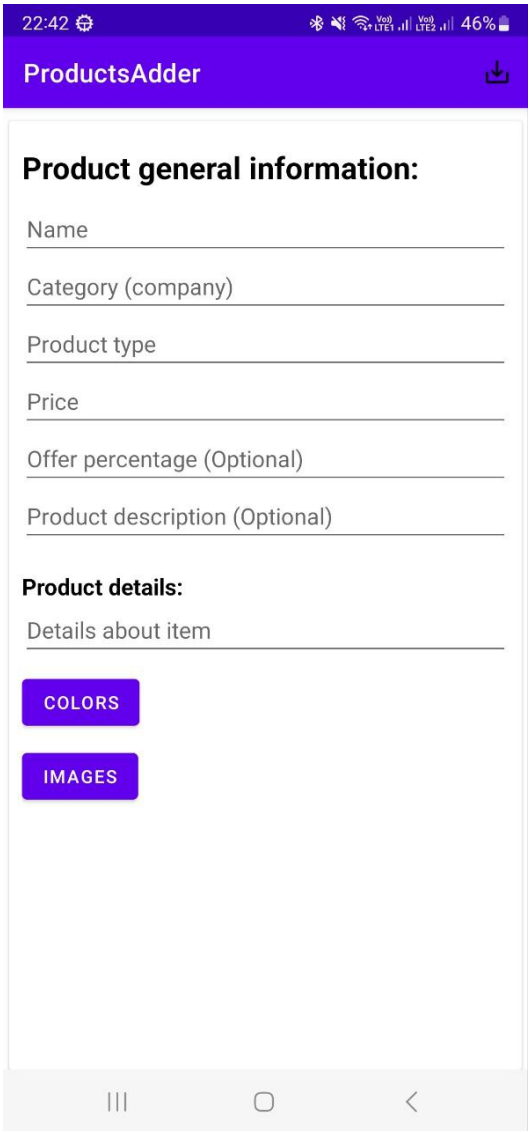


Рисунок 42 - Main Activity

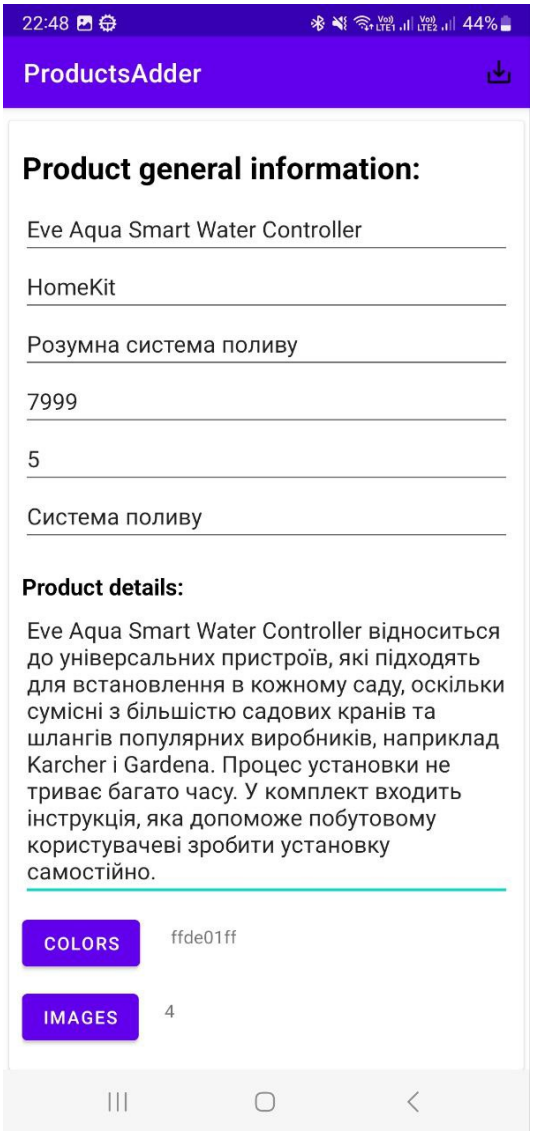


Рисунок 43 - заповнені дані

Після чого, як ми справа зверху в меню настинемо кнопку, спрацює метод saveProduct(), ми отримаємо всі дані з тектового вводу та ці дані буде відправлено в наші сервіси, після чого ми зможемо одразу їх побачити в нашому магазині.

ВИСНОВКИ

Досліджено основні концепції сфери “розумний будинок”. Проаналізовано технології та інструментальні засоби створення мобільних застосунків.

Розроблено мобільний застосунок для ОС Android “Додаток для продажу електроніки в сегменті розумних будинків” у двох версіях - для потенційних покупців та для співробітників компанії з продажу обладнання для “розумного будинку”, у тому числі: сформульовано вимоги до системи, користувацький інтерфейс на базі UX, імплементовано ролі користувача, проведено тестування.

Перспективні напрями розробки:

- Розширення підтримуваних брендів: забезпечити сумісність з більшою кількістю брендів девайсів для розумного будинку.
- Розширення функціональності: додати нові функції, які полегшать пошук бажаних девайсів. Можливість поділитися обраним в магазині девайсом, та можливість залишати користувачам коментарі до пристрою. Також можливість порівняти декілька обраних товарів на одному екрані
- Аналітика та звіти: Додати Google Analytics щоб розуміти чому користувач віддає перевагу та рекомендувати йому необхідні товари які він колись шукав

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. History of Mobile Applications by prof. John F. Clark – University of Kentucky, 2012.
2. Дослідження Statista про прогноз доходу в сегменті розумних будинків [Електронний ресурс] - Режим доступу до ресурсу:https://www.statista.com/forecasts/887687/smart-home-revenue-per-segment-in-the-world
3. Що таке розумний будинок: функції, види, складові та екосистеми [Електронний ресурс] - Режим доступу до ресурсу: https://ek.ua/ua/post/1990/618-chto-takoe-umnyy-dom-funkcii-vidy-sostavlyayuschie-i-ekosistemy/
4. Савонік О. М. Порівняльний аналіз мобільних операційних систем для розробників-початківців / О. М. Савонік, К. О. Скоробагатько. // Університет Григорія Сковороди в Переяславі. – 2021. – No36. - С. 122– 124.
5. Дослідження Statista про популярність ринку мобільних операційних систем у всьому світі 2009-2022 року року [Електронний ресурс] - Режим доступу до ресурсу: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
6. Що таке Git, його особливості [Електронний ресурс] - Режим доступу до ресурсу: https://qagroup.com.ua/publications/shcho-take-git/
7. Gradle Essentials / Kunal Dabir Abhinandan – Packt Publishing Ltd., 2015 – 274 с.
8. Android Developer Fundamentals Course: Practical Workbook / Google Developer Training Team, - Google, 2016 – 566 с.

9. Layout в View [Електронний ресурс] - Режим доступу до ресурсу: https://developer.android.com/develop/ui/views/layout/declaring-layout
10. Створення динамічних списків за допомогою RecyclerView [Електронний ресурс] - Режим доступу до ресурсу: https://developer.android.com/develop/ui/views/layout/recyclerview
11. Короткий посібник з MVVM: ключові моменти та приклади [Електронний ресурс] - Режим доступу до ресурсу: https://highload.today/kratkoe-rukovodstvo-po-mvvm-klyuchevye-momenty-i-primery/
12. Короткий вступ у впровадження залежностей: що це і коли це необхідно використовувати [Електронний ресурс] - Режим доступу до ресурсу: https://xufocoder.medium.com/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-de1367295ba8
13. Що таке Firebase, його особливості [Електронний ресурс] - Режим доступу до ресурсу: https://avada-media.ua/ua/services/firebase/