Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))
Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках»

Виконав: студент 2-го курсу, групи 1ПІ-23м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Карабіньовський Д.М.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ:
Черноволик Г.О.
(прізвище та ініціали)
«06» грудня 2024 р.

Опонент:
к.т.н., доц. каф. ОТ Колесник І.С.
(прізвище та ініціали)
«06» грудня 2024 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н, проф. Романюк О.Н.
(прізвище та ініціали)
«06» грудня 2024 р.

ВНТУ – 2024

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«18» вересня 2024 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙУ РОБОТУ СТУДЕНТУ

Карабіньовському Даниїлу Максимовичу

1. Тема роботи – «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках».

Керівник роботи: Черноволик Галина Олександрівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «17» вересня 2024 р. № 310.

2. Строк подання студентом роботи

3 грудня 2024 року

3. Вихідні дані до роботи: середовище розробки Android Studio, мова розробки Kotlin, операційна система – Windows 10,

4. Зміст текстової частини: вступ; аналіз стану розвитку використання доповленої реальності в андроїд застосунках, розробка методу та моделей Android-системи; розробка Android-системи для використання доповненої реальності в андроїд застосунках; тестування програми; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу: графічний інтерфейс мобільного застосунку; модель роботи мобільного застосунку; блок-схеми алгоритмів роботи мобільного застосунку; тестування мобільного застосунку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Черноволик Г.О., к.т.н., доцент кафедри ПЗ	19.09.24	02.12.24
5	Причепа І.В, к.е.н., доцент кафедри ЕПВМ	22.11.24	30.11.24

7. Дата видачі завдання 19 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку систем доповненої реальності в андроїд застосунках та постановка задач дослідження	20.09.2024 30.09.2024	
2	Розробка методів та моделей Android-системи	01.10.2024 17.10.2024	
3	Розробка програми	18.10.2024 04.11.2024	
4	Тестування програми	05.11.2024 21.11.2024	
5	Економічна частина	22.11.2024 30.11.2024	

Студент _____ Карабіньовський Д.М
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Черноволик Г.О.
(підпис) (прізвище та ініціали)

Анотація

УДК 004.912.032.26

Карабіньовський Д.М. Методи та програмні засоби використання доповненої реальності в модернізованих Андроїд застосунках. Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 166с.

На укр. мові. Бібліогр.: назв: 51; рисунків: 34; таблиць: 14.

У магістерській кваліфікаційній роботі подано результати дослідження технологій розробки Android-додатків із використанням доповненої реальності для підвищення якості користувацького досвіду. Обґрунтовано доцільність використання сучасних методів і засобів для впровадження інтерактивного візуального контенту, що відповідає сучасним вимогам цифрового досвіду.

Магістерська кваліфікаційна робота містить аналіз відомих мобільних додатків, які використовують технології доповненої реальності, особливостей їхньої архітектури та функціоналу, а також методів, що дозволяють забезпечити інтерактивність і високу продуктивність додатків.

Розроблено методи інтеграції елементів доповненої реальності у мобільний додаток, що включають створення 3D-об’єктів і елементів інтерфейсу, а також забезпечення їхньої взаємодії з користувачем. Реалізовано модуль доповненої реальності для відображення віртуальних об'єктів у реальному просторі, розроблено архітектуру класів і сервісів для підтримки інтерактивного функціоналу, а також проведено тестування працездатності та продуктивності системи. Android-додаток розроблено з використанням інтегрованого середовища розробки Android Studio, мови програмування Kotlin та фреймворку Jetpack Compose для створення сучасного графічного інтерфейсу.

Ключові слова: Android-додаток, доповнена реальність, мобільні додатки, 3D-моделювання, тестування продуктивності.

Abstract

Karabinovskyi D.M. Methods and Software Tools for Using Augmented Reality in Modernized Android Applications. Master's Qualification Thesis in the Specialty 121 – Software Engineering, Educational Program – Software Engineering. Vinnytsia: VNTU, 2024. 166 pages.

In Ukrainian. Bibliography: 51 titles; illustrations: 34; tables: 14.

The master's qualification thesis presents the results of research on the development technologies of Android applications using augmented reality to enhance user experience quality. The study substantiates the relevance of modern methods and tools for integrating interactive visual content that meets current digital experience standards.

The thesis includes an analysis of existing mobile applications utilizing augmented reality technologies, focusing on their architectural and functional features, as well as methods to ensure interactivity and high application performance.

Methods for integrating augmented reality elements into mobile applications have been developed, including the creation of 3D objects and interface elements, as well as ensuring their interaction with the user. A module for augmented reality was implemented to display virtual objects in real-world environments. Additionally, the architecture of classes and services supporting interactive functionality was designed, and the system's performance and operability were thoroughly tested.

The Android application was developed using the Android Studio integrated development environment, the Kotlin programming language, and the Jetpack Compose framework for creating a modern graphical user interface.

Keywords: Android application, augmented reality, mobile applications, 3D modeling, performance testing.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Швидкий розвиток сучасних технологій суттєво змінює способи обслуговування клієнтів і підходи до підвищення конкурентоспроможності бізнесу. Це особливо актуально для сфери розробки мобільних додатків [1], де конкуренція є високою, а вимоги користувачів до якості продуктів постійно зростають. Традиційні методи взаємодії з користувачами, які ще нещодавно задовольняли основні потреби аудиторії, вже не відповідають сучасним стандартам і очікуванням. Сьогодні користувачі шукають більше ніж просто функціональність – вони хочуть отримувати якісний цифровий досвід, що включає інноваційні підходи, інтерактивність та персоналізацію.

Однією з технологій, яка активно змінює підходи до створення мобільних додатків, є доповнена реальність (AR). Вона дозволяє інтегрувати в цифрове середовище візуальні, інтерактивні елементи, які роблять процес взаємодії з додатком значно цікавішим і зручнішим для користувачів. Наприклад, AR може використовуватися для демонстрації товарів у тривимірному форматі, створення віртуальних турів, інтерактивного навчання чи навіть розважальних функцій. Такі технології дозволяють не лише розширити функціональні можливості додатків, але й покращити загальний досвід користувачів, створюючи позитивні асоціації та підвищуючи лояльність до бренду.

Сучасні дослідження, проведені в різних країнах, показують, що більше половини користувачів позитивно сприймають можливість використання доповненої реальності у мобільних додатках. Вони відзначають її ефективність для перегляду продукції, навігації в просторі чи взаємодії з інтерфейсом. Наприклад, у галузі електронної комерції AR дозволяє покупцям віртуально приміряти одяг, розміщувати меблі у своєму домі перед покупкою або оцінювати дизайн продуктів у реальному масштабі. У сфері туризму ця технологія може забезпечити віртуальні екскурсії по визначних пам'ятках чи інтерактивні маршрути. В освіті AR дає змогу створювати захоплюючі інтерактивні уроки, які активізують навчальний процес.

Завдяки таким інноваціям мобільні додатки стають не просто засобом для виконання функцій, а повноцінним інструментом для взаємодії з користувачем, який залишає позитивні враження. Тому актуальною є розробка нових методів і підходів до інтеграції технології доповненої реальності в мобільні додатки, що дозволить не лише задовольнити сучасні потреби користувачів, а й створити унікальні рішення, які забезпечать бізнесу конкурентну перевагу.

Зв’язок роботи з науковими програмами, планами, темами.

Дослідження проведено згідно з планом наукових досліджень кафедри програмного забезпечення.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення інтерактивності мобільних Android-додатків шляхом розробки і впровадження методів і засобів використання доповненої реальності, що дозволить розширити функціональні можливості системи.

Для досягнення мети необхідно виконати такі основні завдання:

- визначити методи та засоби реалізації доповненої реальності в мобільних Android-застосунках;
- розробити метод інтеграції доповненої реальності для взаємодії користувача з віртуальними елементами;
- створити програмні модулі для відображення 3D-об'єктів та анімацій;
- розробити метод персоналізованого навчання користувача роботи з доповненою реальністю;
- провести тестування розробленої системи на предмет продуктивності та зручності використання.

Об’єктом дослідження є процес розробки мобільного додатка Android із застосуванням технології доповненої реальності.

Предметом дослідження є методи та засоби реалізації доповненої реальності в Android-застосунках.

Методи дослідження. У процесі дослідження використовувалися такі методи:

- методи розробки мобільних додатків на базі операційної системи Android для реалізації функціоналу доповненої реальності;
- методи об’єктно-орієнтованого програмування для створення архітектури класів і сервісів програмного забезпечення;
- методи 3D-моделювання для створення об’єктів і елементів інтерфейсу;
- методи тестування для перевірки працездатності та продуктивності роботи застосунку.

Наукова новизна отриманих результатів.

1. Подальший розвиток отримав метод інтерактивного відображення даних і взаємодії з користувачем за допомогою доповненої реальності, який, на відміну від існуючих, використовує анімований оверлей, орієнтований на швидке реагування на дії користувача, що сприяє покращенню користувацького досвіду і поліпшить взаємодію користувача з середовищем мобільної системи.
2. Подальшого розвитку отримав метод створення віртуальних елементів та 3D-об’єктів, який, на відміну від існуючих, автоматизує процеси інтеграції об’єктів у віртуальний простір за допомогою додаткового кешування характеристик об’єкту, що дозволяє розширити функціонал Android системи в процесі взаємодії з об’єктами доповненої реальності.

Практичне значення отриманих результатів. Розроблена Android-система з функціями доповненої реальності може застосовуватися для покращення користувацького досвіду в різних сферах, таких як освіта, роздрібна торгівля, туризм, надаючи інтерактивний контент у режимі реального часу.

Особистий внесок здобувача. Усі наукові результати, викладені в магістерській кваліфікаційній роботі, отримані автором особисто. У науковій роботі [2], опублікованій у співавторстві, автору належать аналіз функціоналу мобільних додатків із застосуванням доповненої реальності, постановка задач дослідження, визначення методів їх розв’язання та програмна реалізація основних компонентів системи.

Апробація матеріалів магістерської кваліфікаційної роботи. Результати магістерської кваліфікаційної роботи обговорювалися на XVII

міжнародній науково-практичній конференції «Інформаційні технології і автоматизація – 2024» (Одеса, 2024).

Публікації. Результати дослідження опубліковані в тезах доповіді XVII міжнародної науково-практичної Інтернет-конференції «Інформаційні технології і автоматизація – 2024» (Одеса, 2024) [2].

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку використаних джерел, що містить 35 найменувань, 4 додатки. Робота містить 34 ілюстрації, 14 таблиць.

1 АНАЛІЗ МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ВИКОРИСТАННЯ ДОПОВНЕНОЇ РЕАЛЬНОСТІ В АНДРОЇД ЗАСТОСУНКАХ

1.1 Аналіз стану розвитку доповненої реальності в модернізованих Android-застосунках

Мобільні технології не лише залишаються актуальними, але й стали невід'ємною частиною сучасного життя, суттєво впливаючи на способи спілкування, роботи, навчання та відпочинку. Швидкий розвиток цієї галузі відкриває нові можливості для інновацій та впровадження технологій у різних сферах. Операційна система Android [3] , розроблена компанією Google, є однією з найпоширеніших мобільних платформ у світі. Її популярність обумовлена кількома важливими факторами.

Популярність цієї системи зумовлена такими факторами:

- Android має велику користувацьку базу по всьому світу, мільярди смартфонів і планшетів працюють на цій операційній системі;
- операційна система Android працює на пристроях різних виробників і моделей, що дає користувачам широкий вибір від доступних смартфонів до флагманських моделей, які підходять для різних бюджетів і потреб;
- використовуються постійні оновлення системи від всім нам відомого вендора – Google;
- система має відкрите джерело коду;
- Android пропонує різноманітні функції, включаючи розширену підтримку для розширеної реальності (AR [4]), інтернету речей (IoT), віртуальної реальності (VR), голосового керування та інших інноваційних технологій.

Мобільні додатки з використанням доповненої реальності стрімко набирають популярність і перетворюються на важливу частину сучасного технологічного простору. Вони відкривають нові можливості для користувачів, поєднуючи реальний світ із віртуальними елементами, що дозволяє значно покращити взаємодію з навколишнім середовищем та робить мобільні додатки

ще більш корисними та інтерактивними. Доповнена реальність забезпечує унікальний користувацький досвід у різних галузях, таких як освіта, туризм, шопінг і навіть догляд за здоров'ям.

Одним із головних аспектів використання AR у мобільних додатках є можливість віртуальної взаємодії з об'єктами у реальному часі. Наприклад, у галузі шопінгу користувачі можуть застосовувати додатки, які дозволяють "приміряти" одяг або аксесуари за допомогою камери свого смартфона. Завдяки технології AR, покупці можуть побачити, як виглядатимуть певні речі на них до моменту покупки, що суттєво полегшує процес вибору та знижує ризики неправильного рішення.

Доповнена реальність активно використовується також у сфері освіти. Мобільні додатки з AR пропонують учням і студентам нові, більш інтерактивні форми навчання. Наприклад, додатки для вивчення біології можуть відображати 3D-моделі анатомії людини або тварин, дозволяючи користувачам детально розглянути різні органи і системи організму в об'ємному вигляді. У географії можна взаємодіяти з віртуальними картами або досліджувати 3D-моделі гірських систем чи океанських глибин. Це робить навчання більш захопливим і доступним, оскільки студенти отримують можливість "доторкнутися" до складних концепцій, а не лише читати про них у підручниках.

У туристичній сфері доповнена реальність також пропонує чимало інноваційних рішень. Наприклад, мобільні додатки можуть додавати віртуальні гіди під час відвідування історичних місць або музеїв. Використовуючи камеру смартфона, користувачі можуть отримувати інформацію про різні об'єкти та пам’ятки безпосередньо під час їхнього перегляду. Це може бути у вигляді тексту, аудіо або віртуальних персонажів, які "розповідають" історії про дане місце. Така функціональність дозволяє зануритися в історію і культуру більш інтерактивним способом, ніж звичайний туристичний гід або аудіотур.

AR-додатки в галузі здоров'я надають користувачам можливість моніторити свій фізичний стан, отримуючи віртуальні підказки для тренувань або медичних консультацій. Наприклад, існують мобільні додатки, які

використовують доповнену реальність для надання вказівок під час виконання фізичних вправ. Використовуючи камеру смартфона, користувачі можуть отримувати поради щодо правильної техніки виконання вправ, переглядаючи віртуальні моделі рухів, або відстежувати своє тіло для точнішого виконання вправ. Це робить тренування більш ефективними і безпечними, знижуючи ризик травм.

Доповнена реальність знаходить застосування й у сферах розваг та ігрової індустрії. AR-ігри, такі як знаменитий "Pokémon Go" [5], дозволяють гравцям взаємодіяти з віртуальними персонажами, які з'являються у реальному світі через екран смартфона. Такі додатки поєднують фізичний простір із ігровим досвідом, що робить ігри більш захопливими та занурюючими. Завдяки AR, гравці можуть взаємодіяти з оточенням у нових, унікальних способах, а реальний світ стає частиною ігрового процесу.

Отже, мобільні додатки з використанням доповненої реальності відкривають широкі можливості для інтерактивної взаємодії користувачів із реальним світом. Вони роблять процес навчання, шопінгу, подорожей і навіть догляду за здоров'ям більш зручним, цікавим та інформативним. Завдяки доповненій реальності користувачі можуть отримувати візуальну інформацію безпосередньо з оточення, що дозволяє їм приймати обґрунтовані рішення та покращує загальний досвід взаємодії з технологіями. Розвиток доповненої реальності у мобільних додатках продовжує набирати обертів, і з огляду на швидкий темп цифрової трансформації, цей напрямок має значний потенціал для подальшого зростання в різних сферах життя.

1.2 Порівняння з аналогами

Доповнена реальність (AR) значно змінила мобільний досвід, інтегруючись у різні додатки, які покращують взаємодію користувачів з навколишнім світом через цифрові елементи. На Android платформах ця технологія використовується у багатьох сферах – від розваг до навчання і шопінгу. Порівнюючи різні мобільні додатки, які застосовують AR, можна

помітити, як вони пропонують унікальні способи взаємодії, використовуючи різні методи доповненої реальності для досягнення своїх цілей. Цей аналіз допоможе вивчити, як AR була реалізована в різних додатках і які підходи є найбільш ефективними для тих чи інших задач.

Один із яскравих прикладів використання AR – це додаток IKEA Place [6], який надає можливість користувачам віртуально "розставляти" меблі у власному домі (рис.1.1). За допомогою камери смартфона та доповненої реальності користувачі можуть вибирати меблі з каталогу та розміщувати їх у своїх кімнатах, щоб оцінити, як вони виглядатимуть у реальному просторі. Це дозволяє уникнути непорозумінь при виборі меблів, покращує користувацький досвід і допомагає приймати обґрунтовані рішення перед покупкою. Додаток використовує технологію AR для того, щоб точно позиціонувати віртуальні меблі на поверхнях і надавати реалістичні зображення, враховуючи масштаб і освітлення. Це робить IKEA Place потужним інструментом для тих, хто бажає попередньо спланувати облаштування своєї оселі.

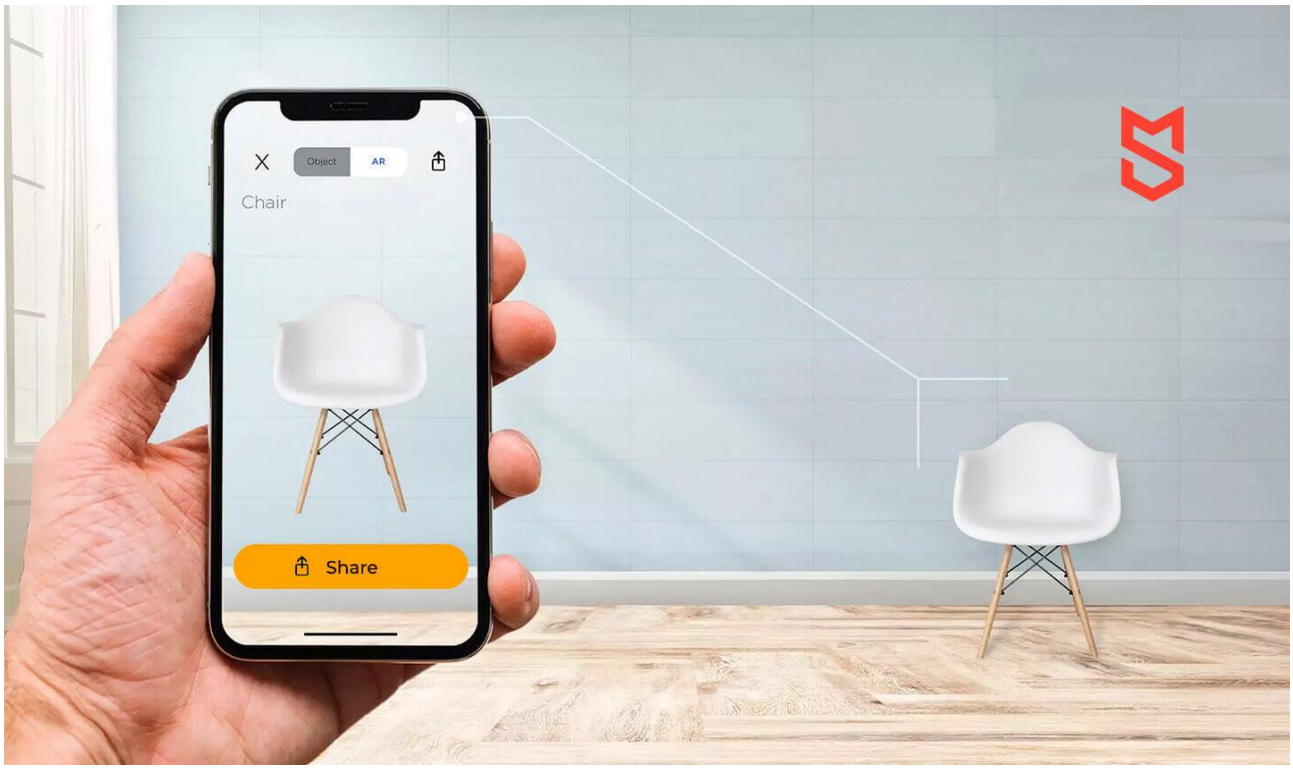

Рисунок 1.1 – Застосунок IKEA Place

На противагу цьому, інший популярний додаток, Google Lens [7], застосовує AR у зовсім іншому контексті – для розпізнавання об'єктів у реальному світі (рис.1.2). Замість того, щоб накладати віртуальні об'єкти на поверхні, Google Lens дозволяє користувачам сканувати об'єкти за допомогою камери свого смартфона, а потім отримувати інформацію про них. Це може бути текст, який автоматично перекладається, квітка, яку можна ідентифікувати, або навіть книга, про яку можна дізнатися більше інформації. В цьому випадку AR використовується як засіб для надання додаткових даних в реальному часі, роблячи взаємодію з оточенням більш інформативною та зручною. Google Lens не фокусується на візуалізації об'єктів, як IKEA Place, але пропонує сильні можливості для розпізнавання та розуміння світу навколо користувача.

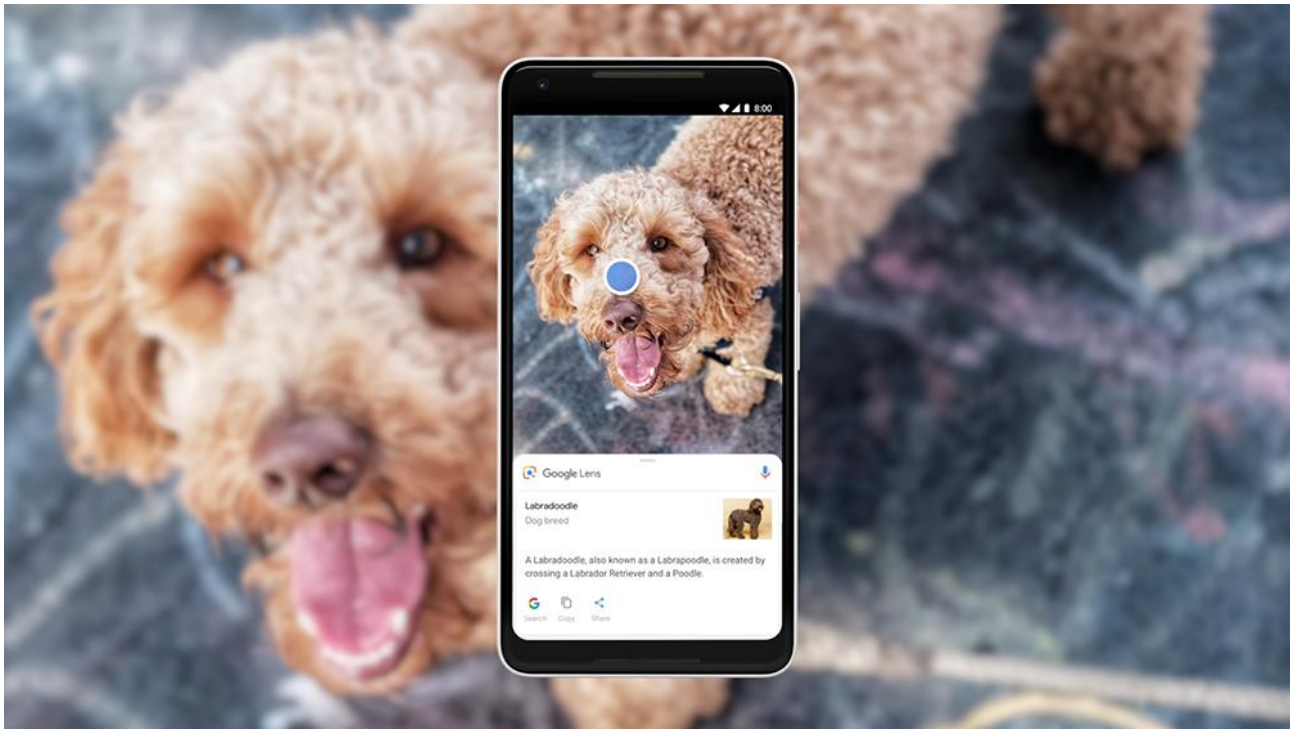

Рисунок 1.2 – Застосунок Google Lens

Ще один приклад – це Snapchat [8], який використовує AR для створення розважального контенту (рис.1.3). Snapchat став відомим завдяки своїм фільтрам, які використовують доповнену реальність, щоб накладати різні анімаційні ефекти або маски на обличчя користувача в режимі реального часу. Ці фільтри

можуть перетворювати обличчя користувачів у різні персонажі, змінювати їхній зовнішній вигляд, додаючи кумедні елементи або навіть анімацію. AR тут використовується для створення інтерактивного та персоналізованого досвіду, який є не лише захопливим, а й інтуїтивно зрозумілим для користувачів. Завдяки своїй популярності, Snapchat став прикладом того, як доповнена реальність може підвищити залученість користувачів і створити нові форми взаємодії в соціальних мережах.

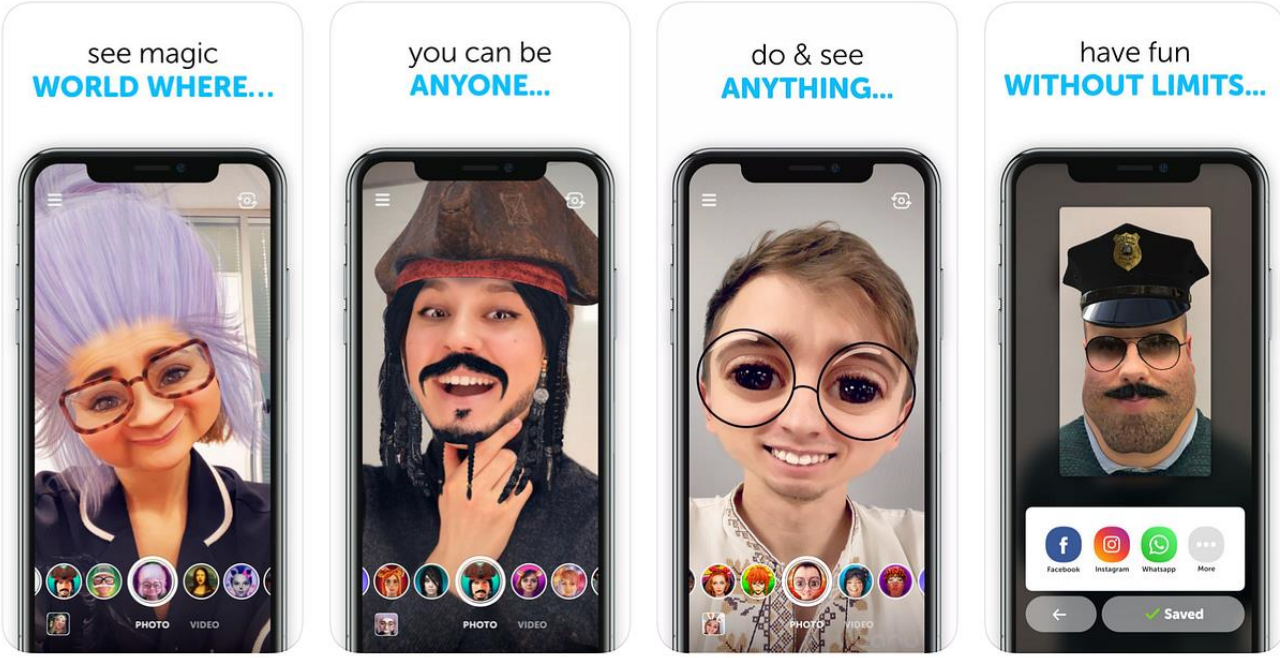

Рисунок 1.3 – Застосунок Snapchat

У сфері освіти доповнена реальність використовується у додатку Quiver [9], який надає дітям можливість бачити оживаючі малюнки (рис.1.4).

Після того, як діти розфарбують малюнок на папері, вони можуть використовувати додаток Quiver для сканування цього малюнка за допомогою камери смартфона. Після цього малюнок оживає у форматі 3D-анімації на екрані. Це робить процес навчання більш захопливим і залучає дітей до активного дослідження через гру та інтерактивність. Використання AR тут допомагає розвивати уяву та творчі навички, перетворюючи статичні зображення на динамічний контент.

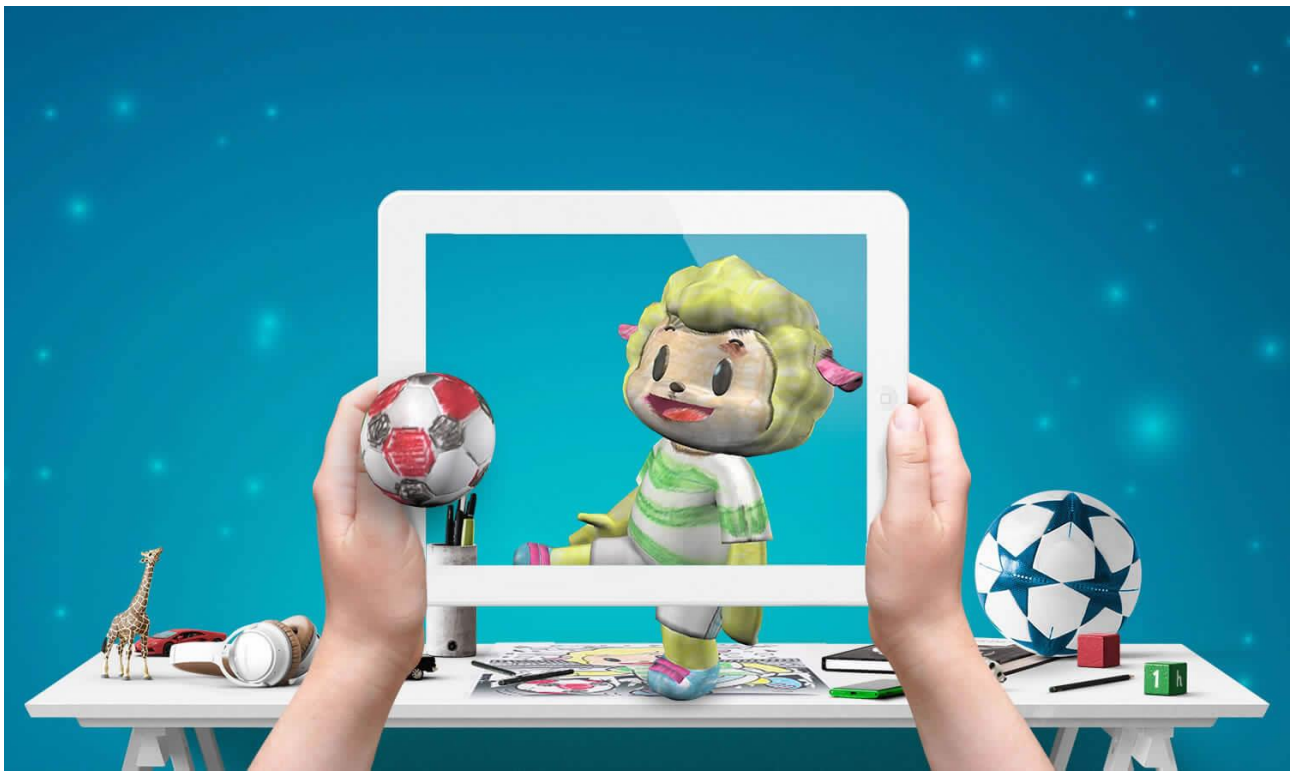

Рисунок 1.4 – Застосунок Quiver

Використовуючи спеціальний сканер, який накладає зображення вен на шкіру пацієнта, AccuVein допомагає медичним працівникам точно визначити місце для ін'єкцій (рис.1.5).

Рисунок 1.5 – Застосунок AccuVein

Це значно підвищує ефективність та точність процедур, знижуючи ризик помилок і дискомфорту для пацієнтів.

Тут AR стає життєво важливою технологією, яка допомагає у медичній діагностиці та лікуванні, надаючи персоналу більше інформації та інструментів для прийняття рішень.

Порівнюючи ці додатки, можна побачити, як різні методи AR використовуються для досягнення різних цілей.

У застосунку IKEA Place акцент робиться на точній візуалізації предметів у реальному просторі, що допомагає користувачам приймати рішення перед покупкою, тоді як Google Lens фокусується на розпізнаванні та наданні інформації про оточуючі об'єкти.

Застосунок Snapchat і його фільтри використовують AR для розваг, накладаючи анімаційні елементи на реальні зображення, що забезпечує інтерактивний і персоналізований досвід.

Застосунок Quiver використовує AR для навчальних цілей, перетворюючи статичні малюнки в анімовані об'єкти, що робить навчання більш динамічним і захопливим для дітей.

Застосунок AccuVein демонструє, як AR може бути використана у сфері охорони здоров'я, наприклад, допомагаючи у проведенні медичних процедур за допомогою точної візуалізації підшкірних вен.

Ці приклади показують, що використання AR в Android-додатках може значно змінити спосіб, яким люди взаємодіють з навколишнім світом, незалежно від сфери – будь-то шопінг, освіта, розваги чи медицина.

Кожен з розглянутих додатків використовує AR по-своєму, залежно від потреб користувача та завдань, які має вирішити застосунок.

Після аналізу усіх аналогів визначено їхні переваги й недоліки та проведено порівняння з власним Android-додатком, який розробляється. Результат порівняння зведено в таблицю 1.1.

Таблиця 1.1 – Порівняльні характеристики мобільних додатків з доповненою реальністю

Назва додатку / Критерій	IKEA Place	Google Lens	Snapchat	Quiver	AccuVein	AR Room
Реалістичність і точність AR-контенту	+	+	-	-	+	+
Наявність персоналізованого ознайомлення	-	-	-	-	-	+
Адаптивність до навколишнього середовища	+	+	+	-	-	+
Сумісність з пристроями	-	+	+	+	-	+
Продуктивність	+	-	-	-	+	+
Підсумковий результат	3	2	2	2	2	5

Отже, можна зробити висновок, що у порівнянні з існуючими аналогами, розробка власного мобільного додатку є доцільною. В результаті роботи отримаємо додаток, який буде покривати недоліки існуючих на даний момент аналогів та забезпечить вищу ефективність роботи.

1.3 Аналіз методів розв’язання задачі

В останні десятиріччя популярність мобільних девайсів все зростає. Мобільні пристрої стали надзвичайно популярними, і їх використання зростає стрімкими темпами. Виробники мобільних телефонів інвестують значні кошти в розробку та вдосконалення технологій, які лежать в основі цих пристроїв. Одним

із найбільш швидкозростаючих напрямків у сфері програмування є створення додатків для смартфонів, оскільки їх кількість перевищує кількість персональних комп'ютерів.

Процес розробки додатків для конкретної платформи або пристрою відомий як нативна розробка, також існують кросплатформені і гібридні додатки, які створюються за допомогою таких фреймворків, як ReactNative [10], Flutter або Xamarin. Нативна розробка передбачає написання програмного забезпечення на специфічній мові програмування для певної операційної системи. Це дозволяє отримати високу продуктивність і свободу в реалізації функцій, однак створення таких додатків потребує більших ресурсів і часу. Наприклад, для розробки під Android використовуються Java або Kotlin[11], а для iOS – Swift. Однією з основних переваг цього підходу є створення інтерфейсу, оптимізованого для конкретної платформи.

До недоліків нативної розробки можна віднести високі витрати на створення і підтримку додатків, а також значний час, необхідний для реалізації програмного забезпечення. Створення додатків для різних операційних систем є складним і ресурсомістким процесом. Альтернативою є розробка кросплатформених додатків за допомогою вебтехнологій, таких як HTML, CSS та JavaScript. Такий підхід дозволяє створювати програму, яка може працювати на кількох платформах одночасно, що економить ресурси, оскільки для цього часто потрібен лише один розробник. Водночас кросплатформені рішення можуть мати певні недоліки, зокрема повільнішу реакцію на дії користувача і зниження загальної продуктивності.

Для створення AR-додатків існує кілька платформ, які спрощують процес розробки та надають базові інструменти для роботи з тривимірними моделями та їх візуалізацією. Однією з найбільш популярних є ARCore [12], платформа від Google, яка дозволяє додаткам розпізнавати об'єкти в реальному світі за допомогою камери смартфона і інтегрувати віртуальні елементи у цей простір. Також важливими інструментами для розробників є Unity у поєднанні з Vuforia,

що дозволяє створювати AR-додатки з розпізнаванням зображень і об'єктів. Такі рішення значно спрощують інтеграцію AR у мобільні додатки.

Для впровадження AR в Android-додатки використовуються такі інструменти, як ARCore , що дозволяє створювати захоплюючі візуальні ефекти та інтерактивні функції на базі Android-пристроїв. ARCoreвикористовує три основні можливості для створення доповненої реальності:

1. Відстеження руху. Це дозволяє додатку визначати положення та орієнтацію телефону відносно об'єктів у реальному світі, що є основою для інтеграції цифрових товарів у реальний простір.
2. Виявлення поверхонь. Завдяки цій функції AR-додаток може знаходити горизонтальні та вертикальні поверхні, на які користувачі можуть розміщувати товари для віртуальної взаємодії.
3. Оцінка освітлення. Відображення цифрових об'єктів стає ще реалістичнішим завдяки можливості адаптувати освітлення віртуальних об'єктів до реальних умов освітлення в навколишньому середовищі.

Розробка доповненої реальності включає кілька ключових етапів. Спочатку здійснюється планування та проєктування, де визначаються вимоги до додатка, його функціональність та тип взаємодії користувача з віртуальними об'єктами. Далі важливо створити модель користувацького досвіду, щоб інтеракція з віртуальними елементами була інтуїтивною та комфортною. Наступним етапом є реалізація функцій просторового розпізнавання та відстеження рухів, що дозволяє додатку точно визначати місце знаходження смартфона і розміщувати віртуальні елементи у фізичному просторі. Завершальними етапами є інтеграція тривимірних моделей, тестування та публікація додатка в магазинах мобільних додатків.

Однак процес розробки AR-додатків має і свої виклики. Серед них — необхідність оптимізації програм для роботи на різноманітних пристроях, адже смартфони Android відрізняються за характеристиками, такими як якість камер, процесори та графічні можливості. Також варто враховувати умови, в яких

користувач буде взаємодіяти з AR-додатком, адже освітлення, рух та стабільність камери можуть впливати на якість роботи доповненої реальності. Незважаючи на виклики, AR у мобільних додатках відкриває величезний простір для інновацій. Вона робить взаємодію з реальним світом інтерактивнішою, дозволяючи користувачам взаємодіяти з віртуальними об'єктами у повсякденному житті. Це підвищує якість користувацького досвіду і робить такі додатки важливим інструментом не лише для розваг, але й для освіти, шопінгу та інших сфер. У майбутньому очікується, що розвиток доповненої реальності стане ще потужнішим, відкриваючи нові можливості для мобільних додатків.

Значну роль у створенні успішного додатку відіграє інтерфейс користувача. Для Android існує технологія Jetpack Compose [13], яка значно спрощує процес створення сучасних інтерфейсів. На відміну від попереднього підходу, який базувався на XML, Compose використовує нову філософію розробки інтерфейсу та управління станами додатку. У Compose розробники описують елементи інтерфейсу через спеціальні функції (composable-функції), де визначаються стан і властивості елементів, а система автоматично генерує потрібний інтерфейс на основі цих налаштувань. Однією з важливих особливостей Compose є використання Slot API, що дозволяє динамічно додавати різні елементи інтерфейсу, такі як текст, кнопки або іконки, у спеціальні контейнери, зберігаючи при цьому гнучкість у дизайні.

Jetpack Compose — це сучасний інструмент для створення користувацьких інтерфейсів на Android, який суттєво спрощує і прискорює процес розробки. Головна відмінність Compose від традиційних методів полягає в декларативному підході, де розробники описують, як має виглядати інтерфейс у певному стані, а система автоматично оновлює його при зміні даних. Це суттєво спрощує побудову динамічних та інтерактивних інтерфейсів, роблячи код більш читабельним та зручним для підтримки.

Compose включає в себе різноманітні компоненти для компонування елементів інтерфейсу. Наприклад, компоненти Box, Row, Column дозволяють гнучко налаштовувати розміщення елементів на екрані. Box — це контейнер, в

якому елементи накладаються один на одного; Row і Column дозволяють розміщувати елементи відповідно по горизонталі або вертикалі. Вони особливо корисні при створенні інтерфейсів з простою лінійною структурою, адже дозволяють швидко створити макет. Для роботи зі списками використовується LazyColumn, яка дозволяє динамічно підвантажувати елементи в міру потреби, що значно підвищує продуктивність додатка при роботі з великим обсягом даних.

Для складніших макетів передбачено ConstraintLayout, який дозволяє детально налаштовувати положення елементів на основі взаємозв'язків між ними. ConstraintLayout забезпечує точне розміщення компонентів за допомогою прив’язок до країв, центрів або інших елементів, що є дуже зручним при створенні нестандартних і складних інтерфейсів. Це робить його потужним інструментом для професійних додатків, що вимагають високого рівня кастомізації.

Scaffold — ще один важливий компонент Compose, що забезпечує типову структуру екрану, наприклад, з AppBar, навігаційною панеллю, плаваючою кнопкою дій тощо. Scaffold спрощує створення стандартних інтерфейсів, типових для Android-додатків, таких як екрани з меню знизу чи зверху. Це дозволяє розробникам швидко налаштувати базовий інтерфейс, не витрачаючи багато часу на повторне створення типових елементів.

Compose також підтримує функцію Preview, яка дозволяє розробникам одразу бачити зміни в інтерфейсі під час його розробки. Це є великою перевагою, оскільки спрощує процес тестування і дає можливість візуально оцінити вигляд кожного компонента окремо. Завдяки Preview, розробники можуть ділити інтерфейс на частини, що значно зменшує обсяг коду та підвищує його зрозумілість.

Ключовим параметром у Compose є modifier, який дозволяє налаштовувати властивості елементів. Modifier забезпечує високий рівень гнучкості: від зміни розмірів, кольору та розташування до додавання тіней, обрамлень або анімацій. Наприклад, за допомогою Modifier можна налаштувати вирівнювання елемента

всередині контейнера, змінити його прозорість, створити ефект натискання. Модифікатори також обмежують налаштування певних властивостей для елементів, які їх не підтримують, що допомагає уникнути помилок та робить код інтерфейсу більш контрольованим.

Для управління станами в Compose використовується MutableState [14], який дозволяє інтерфейсу автоматично оновлюватися при зміні даних. Це дуже важливо при створенні інтерактивних інтерфейсів, адже таким чином додаток завжди відображає актуальні дані. Завдяки MutableState розробник може легко відстежувати і контролювати зміни в інтерфейсі, що дозволяє створювати адаптивні додатки з динамічним контентом. Застосування MutableState допомагає ефективно управляти станом кожного компонента, що робить Compose оптимальним інструментом для побудови додатків з високою взаємодією користувача.

Серед переваг Jetpack Compose перед традиційною розробкою на основі XML — можливість працювати виключно з Kotlin-кодом. Це значно спрощує розробку, оскільки розробникам більше не потрібно переключатися між XML і кодом на Kotlin або Java. У Compose усе відбувається в межах однієї мови, що не тільки спрощує робочий процес, але й знижує ймовірність виникнення помилок. Крім того, оскільки Compose базується на декларативному підході, він дозволяє спрощувати тестування компонентів і робить код більш компактним і зрозумілим.

Ще одна важлива особливість — миттєвий перегляд змін у коді завдяки функції Preview[15], яка значно скорочує час розробки. Розробник може бачити зміни в інтерфейсі одразу, що дозволяє швидко тестувати різні ідеї та знаходити оптимальні рішення. Preview дозволяє легко налаштовувати компоненти, робити дрібні правки та бачити, як вони впливають на загальний вигляд інтерфейсу.

Попри численні переваги, використання Compose у великих проєктах може бути викликом. Перехід з XML-розробки на декларативний підхід Compose потребує адаптації. У великих проєктах це може зайняти час, оскільки кожен

елемент інтерфейсу потрібно налаштовувати заново, а також адаптувати процеси розробки під нові методи управління станом та створення динамічного контенту. Однак, враховуючи швидкий розвиток Compose та активну підтримку від Google, з часом цей інструмент стає все більш доступним та зручним для великої кількості проєктів.

Jetpack Compose також включає в себе багато інструментів для анімацій та налаштувань динамічних елементів, які спрощують створення візуально привабливих і сучасних додатків. Вбудовані можливості для анімацій дозволяють плавно змінювати властивості елементів, такі як розмір, колір, положення, роблячи додаток більш інтерактивним та естетично привабливим. Завдяки гнучкому та інноваційному підходу до створення інтерфейсів, Jetpack Compose поступово стає новим стандартом розробки на платформі Android. З його допомогою розробники можуть створювати складні, динамічні інтерфейси з мінімальними зусиллями, що дозволяє значно оптимізувати процес роботи та підвищити якість кінцевого продукту.

Популярність Compose продовжує зростати, і великі компанії, такі як Airbnb, Booking, Lyft, Twitter, вже активно використовують цю технологію. Очікується, що з часом ще більше бізнесів перейдуть на Compose, оскільки він спрощує розробку, надає більше гнучкості та зменшує кількість необхідного коду. Розробники Android все частіше використовують Compose для нових проєктів, і знання цієї технології стане важливою вимогою на ринку праці у майбутньому.

1.4 Постановка задач дослідження

Провівши аналіз стану розвитку Android-систем доповненої реальності, було визначено задачі, які необхідно вирішити у процесі розробки програмного продукту:

- розробити методи і модель роботи системи;
- розробити метод відображення 3D-об'єктів;
- розробити метод персоналізованого навчання користувача роботи з

доповненою реальністю;

− реалізувати метод перегляду 3-D об’єктів в режимі доповненої реальності;

− розробити метод персоналізованого навчання користувача роботи з доповненою реальністю;

− розробити зручний та зрозумілий графічний інтерфейс;

− провести тестування Android-системи.

1.5 Висновки

У першому розділі було розглянуто стан розвитку мобільних систем з використанням доповненої реальності. Було проведено порівняння таких аналогів як IKEA Place, Google Lens, Snapchat, Quiver та AccuVein.. Результати аналізу вказують на недостатність наявного функціоналу для роботи з доповненою реальністю. Було проаналізовано способи розробки мобільних додатків, у результаті аналізу та порівняння переваг і недоліків було обрано метод «нативної розробки». Також було проаналізовано способи написання графічного інтерфейсу XML і Jetpack Compose, в результаті аналізу та порівнянь переваг та недоліків було обрано Jetpack Compose. Було сформульовано основні задачі, які потрібно вирішити при розробці мобільної системи під платформу Android.

2 РОЗРОБКА МЕТОДІВ ТА МОДЕЛЕЙ СИСТЕМИ

2.1 Архітектурне проєктування програмної системи

При розробці мобільного Android-додатку було використано один з найпопулярніших шаблонів програмування Model-View-ViewModel, або скорочено MVVM [16]. Цей шаблон програмування використовується для розробки користувацького інтерфейсу і розділяє між собою бізнес логіку та логіку відображення. Схема взаємодії Model, View і ViewModel у мобільному додатку «AR Room» зображено на рисунку 2.1.

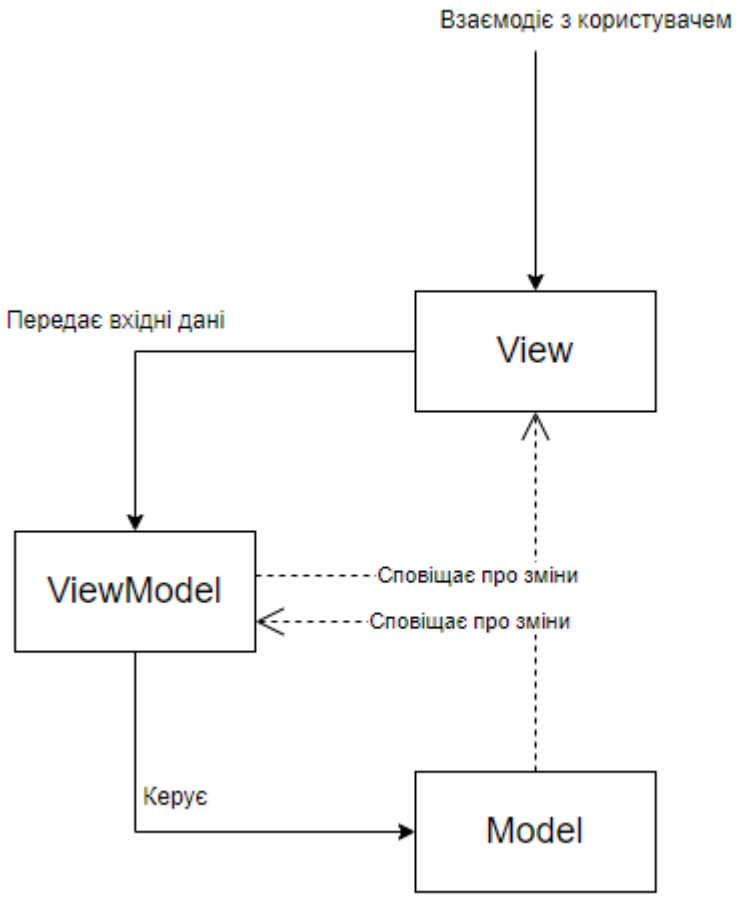

Рисунок 2.1 – Схема взаємодії Model, View і ViewModel у шаблоні MVVM

MVVM (Model-View-ViewModel) — це архітектурний шаблон, який використовується для розділення бізнес-логіки, логіки відображення та обробки даних у додатках, особливо популярний у розробці Android-додатків. Цей підхід дозволяє забезпечити чіткий розподіл відповідальності між компонентами,

підвищити масштабованість додатка, а також полегшити тестування та підтримку.

Архітектура MVVM включає три основні компоненти: модель (Model), представлення (View) та модель представлення (ViewModel). Модель займається управлінням даними, реалізацією бізнес-логіки та взаємодією з джерелами даних, такими як локальні бази даних чи зовнішні API. Наприклад, вона може містити інформацію про список замовлень, отриманий із серверу, або виконувати перевірку введених користувачем даних.

Представлення відповідає за відображення даних користувачеві та обробку його дій, таких як кліки чи введення тексту. Цей компонент не містить складної логіки і лише відображає інформацію, отриману від ViewModel. Наприклад, список товарів може бути показаний у вигляді карток або таблиць, залежно від потреб інтерфейсу.

Модель представлення є посередником між View і Model, забезпечуючи передачу даних між ними. ViewModel отримує дані з моделі, обробляє їх за необхідності та передає у зручному для відображення форматі представленню. Важливо, що ViewModel не має прямої залежності від View, що дозволяє зберігати його незалежність і спрощує підтримку коду. Наприклад, ViewModel може конвертувати дату з моделі у формат, зручний для локального відображення.

Перевагою MVVM є чітке розділення відповідальності між компонентами. Завдяки цьому розподілу розробникам легше орієнтуватися у коді, розуміючи, який із компонентів відповідає за бізнес-логіку, обробку даних чи відображення інтерфейсу. Це також сприяє ефективній командній роботі, адже окремі команди можуть зосередитися на різних аспектах застосунку. Крім того, MVVM спрощує тестування, оскільки логіка бізнес-процесів та обробка даних ізольовані у Model та ViewModel. Це дозволяє перевіряти правильність роботи програми без необхідності взаємодії з інтерфейсом користувача.

Підтримка архітектури в Android Studio робить її ще більш привабливою для використання. Інструменти, такі як LiveData та ViewModel з бібліотеки

Jetpack, полегшують синхронізацію даних і реалізацію реактивного підходу. Це також сприяє зручному управлінню станом інтерфейсу, що позитивно впливає на швидкість розробки.

Втім, MVVM має і свої недоліки. Наприклад, новачкам може бути складно зрозуміти концепцію і впровадити її у складних проєктах, де присутня велика кількість взаємодій між компонентами. Реалізація MVVM потребує більше часу і додаткового коду, оскільки для кожної нової функції потрібно створювати класи для Model, ViewModel і відповідним чином модифікувати View. У випадках, коли принципи MVVM не дотримуються належним чином, може виникнути проблема перевантаження ViewModel, що ускладнює її підтримку та тестування.

Незважаючи на ці виклики, MVVM є потужним інструментом для організації розробки, особливо у масштабних проєктах, де чітке розділення відповідальності та можливість незалежного тестування мають вирішальне значення.

Відзначимо також переваги MVVM у поєднанні з Jetpack Compose:

1. Простота інтеграції. Дані з ViewModel можна безпосередньо використовувати у Jetpack Compose через State чи LiveData, що мінімізує кількість коду. Автоматичне оновлення UI при зміні стану у ViewModel значно спрощує синхронізацію даних.
2. Підвищена продуктивність розробки. Jetpack Compose спрощує роботу над View, а MVVM забезпечує структурований підхід до обробки даних. Розробники можуть зосередитися на логіці та дизайні без необхідності писати зайвий код для зв’язку між компонентами.
3. Масштабованість та зручність підтримки. Завдяки розподілу відповідальності, зміни в бізнес-логіці або інтерфейсі не впливають на інші компоненти. Легко додавати нові функції, використовуючи вже існуючі ViewModel та Model.

Jetpack Compose — це сучасний декларативний фреймворк для створення користувацьких інтерфейсів в Android-додатках, який дозволяє швидко та

ефективно створювати динамічні UI без використання XML-макетів. Compose побудований на мові Kotlin і використовує декларативний синтаксис, що значно спрощує процес розробки, зменшує кількість коду й дозволяє створювати інтуїтивно зрозумілі інтерфейси.

Переваги Jetpack Compose:

- розробники описують не послідовність дій, а стан інтерфейсу, який система автоматично відтворює;
- наприклад, замість того, щоб вручну оновлювати текст чи кольори, програміст вказує умови, за яких ці зміни мають відбутися;
- завдяки використанню Kotlin розробка інтерфейсів стає інтуїтивною, навіть для новачків;
- зменшення обсягу коду на 30% дозволяє зосередитися на функціоналі, а не на написанні складних конструкцій;
- Jetpack Compose дозволяє створювати анімації та інтерактивні елементи без складного коду;
- користувацький досвід значно покращується завдяки миттєвій реакції інтерфейсу на дії користувача;
- компоненти, створені за допомогою Compose, легко переносити між екранами або навіть додатками, що зменшує час розробки для схожих проєктів;
- можна створити власну бібліотеку елементів інтерфейсу, яку згодом використовуватимуть різні команди;
- Compose дозволяє реалізовувати складні макети та анімації, налаштовуючи їх під потреби проєкту;
- наприклад, програміст може створити унікальний вигляд кнопок чи переходів між екранами без залучення додаткових бібліотек;
- інтеграція з Kotlin спрощує написання тестів, дозволяючи автоматично перевіряти працездатність інтерфейсу;
- можна створювати тести для перевірки коректного відображення інтерфейсу на різних екранах і при різних станах додатка.

Недоліки Jetpack Compose:

- це може бути проблемою для команд, які використовують Java чи інші мови програмування, оскільки їм потрібно додатково вивчати Kotlin;
- деякі сторонні бібліотеки, написані на інших мовах, можуть вимагати адаптації для роботи з Compose;
- через динамічний характер рендерингу інтерфейсу можуть виникати лаги, особливо в складних додатках або при використанні анімацій;
- для вирішення таких проблем необхідно оптимізувати компоненти й мінімізувати кількість рекомпозицій;
- якщо додаток уже створено з використанням XML-макетів, його переведення на Jetpack Compose потребує значних витрат часу та ресурсів;
- це особливо складно для великих проєктів із розгалуженою структурою.

Firebase обрано як серверну платформу для мобільного додатку завдяки його багатофункціональності, простоті інтеграції та масштабованості. Firebase пропонує потужні інструменти для зберігання й обробки даних у хмарі, забезпечує високий рівень безпеки та дозволяє швидко розгортати серверну частину. Це ідеальне рішення для мобільних додатків, які потребують надійного серверного середовища для роботи з даними користувачів.

2.2 Розробка методу відображення 3D-об'єктів

У сучасних мобільних додатках відображення 3D-об'єктів стає невід’ємною частиною інноваційних рішень, особливо в областях розширеної реальності, ігор, візуалізації даних та архітектури. Основна складність полягає в тому, щоб забезпечити оптимальну продуктивність і якість графіки, зважаючи на обмеження мобільних пристроїв, такі як процесорна потужність, обсяг оперативної пам’яті та енергоспоживання.

Розробка методу відображення 3D-об’єктів, який враховує всі ці фактори, є завданням, яке вимагає детального підходу. Такий метод має включати функції завантаження моделей, налаштування візуальних ефектів, інтерактивності та анімацій, а також оптимізацію для забезпечення плавного досвіду користувача.

У цьому розділі детально розглядається запропонований підхід до реалізації методу, який спрямований на створення гнучкої, масштабованої та ефективної системи рендерингу 3D-об’єктів.

Розроблений метод повинен забезпечувати:

- створення базового рендерингового середовища для роботи з 3D-графікою;
- завантаження та обробка моделей у популярних форматах, таких як GLTF[17], OBJ або FBX;
- забезпечення візуальних ефектів, включаючи налаштування матеріалів, текстур, освітлення та тіней;
- реалізацію інтерактивності для маніпуляції об’єктами (обертання, масштабування, переміщення);
- Оптимізація продуктивності для плавного відображення навіть на пристроях із середніми технічними характеристиками.

Ці завдання формують основу методу, який дозволяє інтегрувати 3D-об'єкти в мобільний додаток та надає можливість для подальшого розширення функціоналу.

Метод побудований на основі декількох компонентів, які забезпечують його модульність, простоту використання та адаптацію до різних сценаріїв.

Основою для відображення графіки є рендерингова сцена, яка створює 3D-простір для роботи з об’єктами. Вона відповідає за обчислення перспективи, розташування камер, обробку освітлення та інших важливих параметрів.

Завантаження моделей є критичним етапом, оскільки якість та формат вихідних даних впливають на продуктивність. Реалізація підтримує завантаження як локальних, так і віддалених ресурсів із попередньою конвертацією даних у внутрішній формат.

Додаються матеріали, текстури, джерела світла і тіні, які дозволяють підвищити реалістичність об’єктів та їх взаємодію з віртуальним середовищем.

Користувачі повинні мати можливість взаємодіяти з 3D-об'єктами через сенсорний інтерфейс: масштабування, обертання, переміщення, натискання тощо.

Візуальні елементи мають бути динамічними, що досягається через додавання анімацій, таких як рух об’єктів по траєкторії чи зміна властивостей у реальному часі.

Останнім, але не менш важливим компонентом є оптимізація, яка забезпечує збалансоване використання ресурсів пристрою, дозволяючи реалізувати якісний досвід для користувачів із мінімальними затримками.

Клас CustomScene створює базову рендерингову поверхню з можливістю оновлення графіки.

Цей клас забезпечує основу для рендерингу OpenGL, який використовується для відображення об'єктів.

```
class CustomScene(context: Context) : GLSurfaceView(context) {
    private val renderer: SceneRenderer

    init {
        // Установлення OpenGL ES 2.0
        setEGLContextClientVersion(2)
        renderer = SceneRenderer()
        setRenderer(renderer)
        renderMode = RENDERMODE_CONTINUOUSLY
    }
}
```

Рисунок 2.2 – Приклад створення кастомної сцени

Реалізація методу завантаження моделі включає обробку файлів формату

```
fun loadModel(context: Context, filePath: String): Model {
    val assetManager = context.assets
    val inputStream = assetManager.open(filePath)
    return ModelLoader.load(inputStream) // Перетворення у внутрішній формат
}
```

Рисунок 2.3 – Приклад завантаження моделі з пам’яті додатку

Виконання методу для відображення 3D об'єктів (рис. 2.4) складається з таких кроків:

1. Створюється AR-двигун завантажувач 3D-моделей modelLoader.
2. Ініціалізуються об'єкти для роботи з камерою cameraNode та списком childNodes, де будуть зберігатися всі 3D-елементи.
3. Завантажується модель за допомогою шляху, вказаного у arViewUiState.modelPath.
4. Після завантаження модель обертається у вузол ModelNode, її масштаб та початкове обертання встановлюються до базових значень.
5. Камера використовує методи відстеження, щоб знаходити площини у просторі. Якщо модель ще не розміщена, створюється новий "якір" у центрі знайденої площини, до якого додається модель.
6. Додається обробка жестів, таких як одиночний тап для розміщення моделі чи рух для переміщення об'єктів у просторі.
7. Переміщення моделі реалізується через зміну положення "якоря", до якого вона прив'язана.

Рендеринг відбувається через виклик методу ARScene, який малює всі вузли, моделі, камеру та ефекти у вказаному контейнері. Розроблений метод для відображення 3D-об’єктів дозволяє створювати динамічний та інтерактивний інтерфейс у мобільних додатках. Використання таких підходів сприяє вдосконаленню графічної складової додатків, покращенню користувацького досвіду та відкриває нові можливості для інноваційного застосування 3D-графіки.

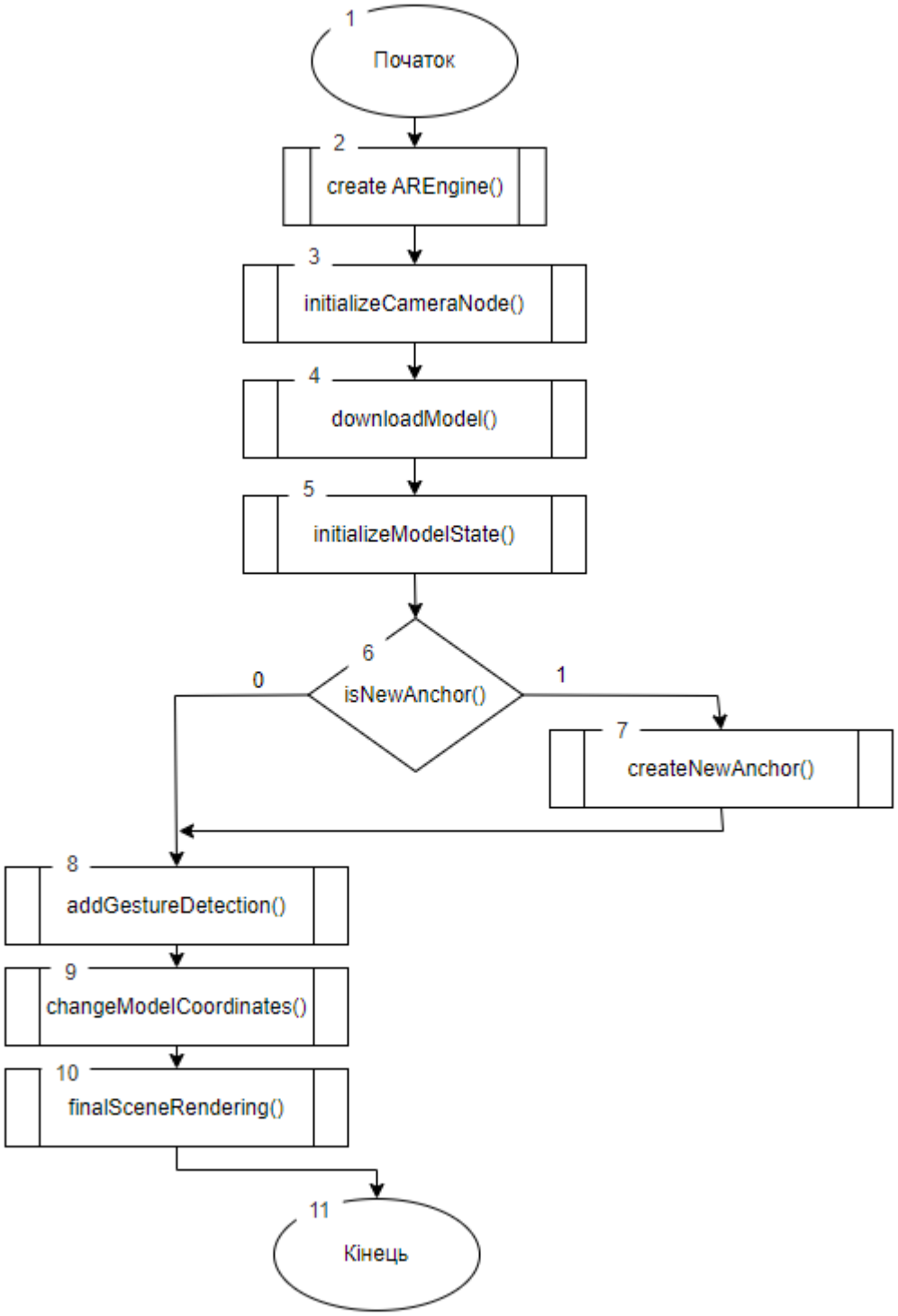

Рисунок 2.4 – Алгоритм методу для відображення 3D об’єктів

2.3 Розробка моделі інтерфейсу з можливістю перевикористання в сторонніх сервісах

У сучасній розробці мобільних додатків все більше уваги приділяється повторному використанню програмних компонентів. Цей підхід дозволяє скоротити час розробки, підвищити якість коду та забезпечити гнучкість

інтеграції між різними проєктами. У рамках цього дослідження пропонується створити модель інтерфейсу, яка легко інтегрується в сторонні проєкти у вигляді бібліотеки формату .aar [18]. Такий підхід дозволить іншим розробникам підключати функціонал вашого додатку до своїх проєктів без необхідності складної кастомізації чи зміни коду.

Створення гнучкої архітектури інтерфейсу передбачає, що:

- компоненти повинні бути простими у підключенні, з чіткою документацією та інтуїтивним API;
- кожен компонент має виконувати окрему задачу і не залежати від інших, що забезпечить зручність повторного використання;
- інтерфейс має працювати у будь-якому Android-додатку незалежно від структури проєкту;
- використання сторонніх бібліотек має бути зведено до мінімуму, щоб уникнути конфліктів;
- інтегратор повинен мати змогу налаштувати компоненти під свої потреби, не змінюючи основного коду.

Серед основних концепцій розробки виділимо:

1. Абстрактність компонентів. Компоненти розробляються таким чином, щоб вони взаємодіяли з зовнішнім середовищем через чітко визначені інтерфейси. Наприклад, кнопки, віджети або екрани не повинні залежати від конкретної реалізації додатку, а використовувати загальні абстракції.

2. Упаковка у вигляді бібліотеки .aar. Формат .aar дозволяє зібрати всі необхідні ресурси, включаючи код, графічні елементи та залежності, в єдиний файл. Це робить інтеграцію максимально простою: розробнику достатньо підключити файл до свого проєкту та використовувати доступний функціонал.

3. Використання ресурсів. Інтерфейсний дизайн має бути побудований на основі стилів і тем, які легко налаштовуються. Це дозволяє інтегратору адаптувати вигляд компонентів до стилю свого додатку.

4. Документоване API. Розробка включає чіткий опис кожного методу, атрибуту та можливості компонента, щоб сторонні розробники швидко

зрозуміли принципи роботи і змогли легко налаштувати функціонал.

Також в ході проєктування було зроблено акцент на таких архітектурних принципах:

1). Розділення відповідальностей. Основний принцип проєктування — розділення відповідальностей [19]. Наприклад, якщо додаток відображає 3D-об'єкти або працює з певними даними, функції для відображення та обробки мають бути відокремлені.

UI-компоненти відповідають лише за відображення інформації.

Бізнес-логіка виконує обробку даних і викликає функції інтерфейсу через абстрактний рівень.

Менеджери ресурсів працюють із завантаженням моделей, текстур або інших даних, забезпечуючи легку зміну джерел даних.

2). Гнучка кастомізація. Щоб забезпечити можливість перевикористання компонентів у різних середовищах, архітектура має дозволяти просту кастомізацію (рис. 2.5). Наприклад, якщо компонент — це кнопка, користувач повинен мати змогу легко змінювати її стиль, розмір або текст.

Припустимо, є компонент для відображення інтерактивного списку. Його реалізація будується навколо базового класу, який надає мінімальний набір функцій.

```
class ReusableListView(context: Context, attrs: AttributeSet?) : RecyclerView(context, attrs) {
    fun setData(items: List<String>) {
        adapter = Adapter(items)
    }
    private fun Adapter(items: List<String>): Adapter<*>? {
        return Adapter(emptyList())
    }
    fun setOnItemClickListener(listener: (position: Int) -> Unit) {
    }
}
```

Рисунок 2.5 – Приклад реалізації гнучкої кастомізації

Цей клас дозволяє інтегратору легко підключати списки до своїх проєктів, задавати дані та реагувати на взаємодію з елементами.

Після створення бібліотеки .aar розробник підключає її до свого проєкту

тадодає .aar файл до модулів (рис. 2.6).

Серед переваг підходу можна виділити:

- сторонні розробники отримують готові компоненти, які не потребують додаткової обробки;
- перевикористання компонентів дозволяє уникнути дублювання роботи;
- компоненти адаптуються до будь-якого стилю чи вимог додатку;
- бібліотека легко розширюється новими функціями;
- нові версії бібліотеки можна швидко розповсюджувати серед користувачів, забезпечуючи їх доступ до актуального функціоналу.

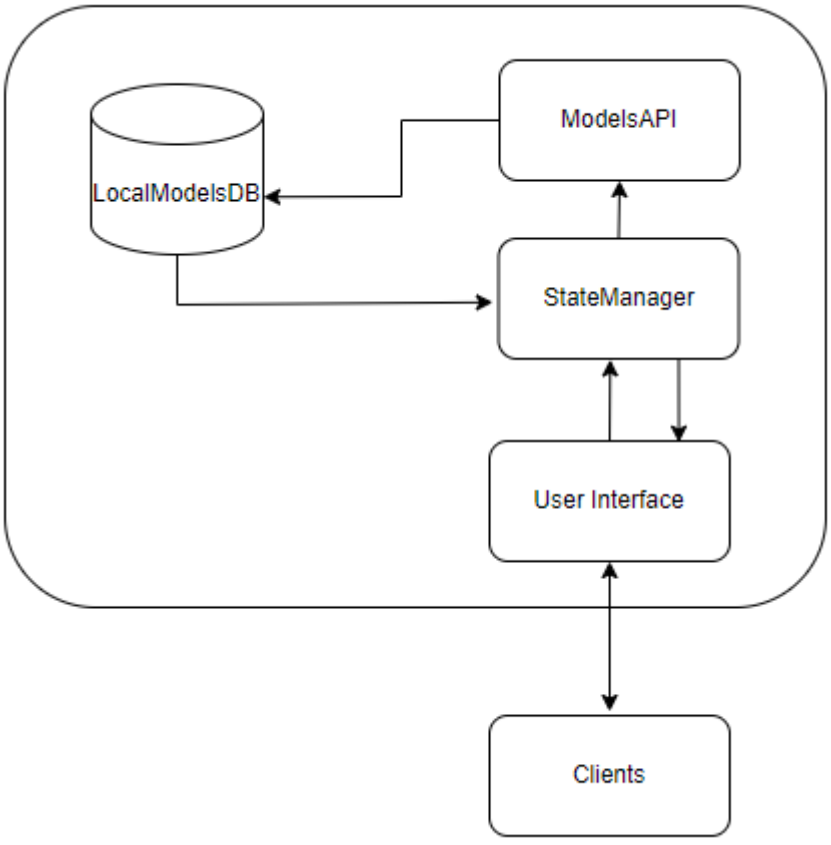

Рисунок 2.6 – Модель інтерфейсу з можливістю перевикористання

Розробка інтерфейсів із можливістю перевикористання у вигляді бібліотеки формату .aar відкриває широкі можливості для спрощення інтеграції функціоналу в інші проєкти. Такий підхід дозволяє не лише підвищити ефективність розробки, але й сприяє створенню екосистеми інноваційних

рішень, які можуть використовуватись у різних галузях.

2.4 Розробка методу персоналізованого навчання користувача роботи з доповненою реальністю

Робота з доповненою реальністю (AR) може бути складною для користувачів, які не мають попереднього досвіду. Персоналізоване навчання дозволяє адаптувати процес освоєння інструментів AR відповідно до індивідуальних потреб, рівня знань і поведінки кожного користувача. У цьому розділі представлено метод розробки персоналізованого навчання, яке забезпечує легкий та ефективний старт роботи з додатками доповненої реальності.

Персоналізація навчального процесу включає адаптацію контенту та функціоналу до індивідуальних потреб користувача. Основні принципи персоналізації:

1) додаток визначає, наскільки добре користувач розуміє функціонал AR;
2) користувач отримує інструкції та навчальні матеріали відповідно до свого рівня досвіду;
3) система аналізує дії користувача та коригує навчальний процес у реальному часі;
4) залучення користувача через інтерактивні завдання, демонстрації та зворотний зв'язок.

Серед компонентів методу можна виділити такі:

1. Стартова анімація пояснює основну концепцію доповненої реальності. На цьому етапі. Віртуальний 3D-об'єкт плавно з'являється на екрані з текстовим поясненням, як його розмістити у просторі.
2. Навчальний оверлей накладається поверх 3D-сцени, підсвічуючи важливі області чи функції. Виглядає як напівпрозорий круг або прямокутник виділяє активну зону для взаємодії (наприклад, кнопки або об'єкт).
3. Демонстрація маніпуляцій з об’єктами заключається в додаванні анімації переміщення об'єкта за допомогою імітованого жесту перетягування.

4. Анімація полягає в тому, що два віртуальні пальці "розтягують" або "стискають" об’єкт для демонстрації масштабування.
5. Додане текстове повідомлення з стрілкою, що вказує на об’єкт чи зону.
6. Система фіксує успішні маніпуляції і підтверджує їх через вимкнення підказок.

Серед особливостей методу виділимо:

- анімації, які адаптуються до поточного контексту сеансу;
- збереження мінімалізму і фокус на ключових функціях.

Для реалізації було використано ARCore. Використання бібліотеки ARCore дозволяє забезпечити точне розпізнавання простору та інтеграцію об'єктів у реальний світ. Також завдяки Завдяки Jetpack Compose забезпечується динамічне відображення навчальних елементів інтерфейсу. DataStore для збереження прогресу. Прогрес навчання зберігається в DataStore, що дозволяє продовжити з того місця, де користувач зупинився.

Запропонований метод (рис. 2.7) включає функціонал:

1) визначення початкового рівня користувача (новачок чи досвідчений);
2) запуск стартової анімації;
3) послідовна демонстрація базових функцій через оверлей;
4) виконання користувачем завдань із реакцією системи на кожен крок;
5) поступовий перехід до складніших функцій;
6) завершення навчання.

Виконання методу складається з таких кроків:

1) задавання анімації жестів через LottieCompositionSpec::RawRes;
2) визначення відповідних текстових повідомлення для кожного жесту через stringResource;
3) використовується змінна playedCount, яка зберігається з можливістю відновлення стану через rememberSaveable;
4) розраховування індексу поточного жесту (currentGestureIndex) залежно від кількості програних анімацій;
5) якщо кількість ітерацій для кожного жесту перевищує задану кількість,

викликається метод onComplete;

6) використовується rememberLottieComposition для зберігання стану анімації;
7) коли прогрес анімації досягає кінця ANIM_COMPLETE_VALUE, викликається функція onEnd, яка збільшує playedCount;
8) анімації повторюються для кожного жесту до досягнення потрібної кількості ітерацій.

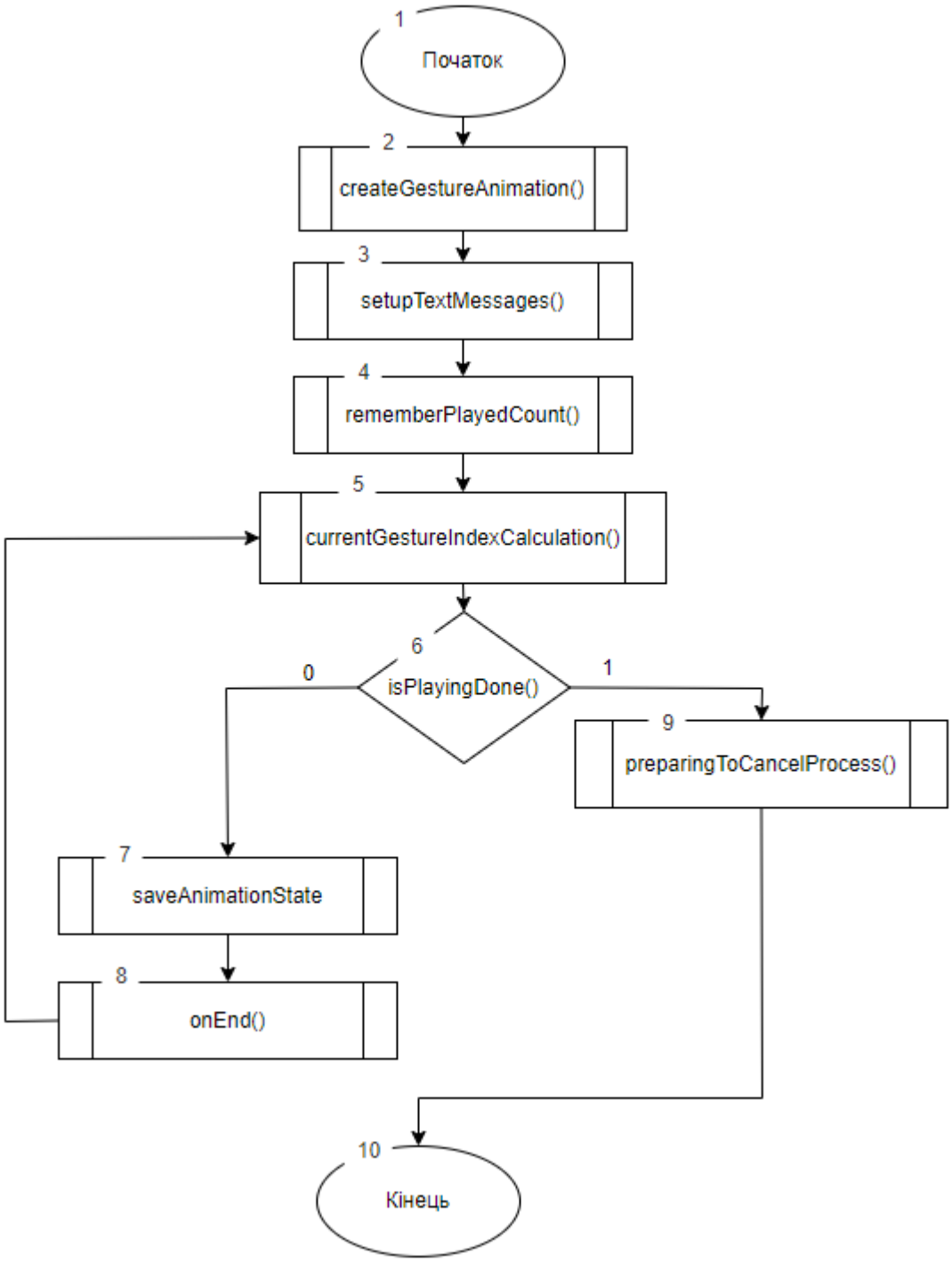

Рисунок 2.7 – Алгоритм методу персоналізованого навчання користувача

Персоналізоване навчання користувачів роботи з доповненою реальністю є важливим елементом для створення інтуїтивного та привабливого додатку. Воно дозволяє адаптувати навчальні процеси під індивідуальні потреби, забезпечуючи комфортне і продуктивне освоєння технології AR.

2.5 Розробка моделі Android - системи

Існування зв’язків між елементами системи та їхня взаємодія є фундаментальною характеристикою будь-якої системи, яка забезпечує її цілісність і функціональність. Такі зв’язки дозволяють визначити, як елементи системи співпрацюють один з одним для виконання спільної мети, забезпечуючи системі її призначення. Аналіз і проєктування цих зв’язків є важливими етапами створення будь-якого програмного продукту, а графічне подання допомагає зробити цей процес більш прозорим і зрозумілим.

Для графічного подання структури, поведінки та функціональності систем використовується уніфікована мова моделювання (UML), яка стала стандартом у розробці програмного забезпечення. UML не є мовою програмування в традиційному розумінні, а виступає набором правил і стандартів, що дозволяють візуально відображати взаємозв’язки в системі через діаграми. Це забезпечує можливість розробникам не лише бачити загальну картину системи, а й глибше розуміти її внутрішню логіку.

Основною перевагою UML [20] є її універсальність і багатогранність, які роблять її корисною у різних галузях. У сфері розробки програмного забезпечення UML сприяє опису архітектури коду, аналізу функціональних можливостей системи та виявленню потенційних проблем ще на стадії проєктування. У бізнес-аналітиці UML дозволяє моделювати процеси, описувати потреби компаній і розробляти рішення, які підвищують ефективність роботи бізнесу. У системному проєктуванні UML застосовується для розробки складних технічних систем, таких як мережеві інфраструктури, автоматизовані системи керування та системи реального часу.

Використання UML дозволяє охоплювати всі аспекти розробки

програмного забезпечення, починаючи від концепції і закінчуючи реалізацією. Ця мова моделювання забезпечує можливість створювати графічні діаграми, які відображають різні аспекти системи, включаючи її структуру, поведінку та фізичну реалізацію. Моделювання структури дає змогу виявити статичні зв'язки між елементами, такі як їхні атрибути, відносини і залежності. Моделювання поведінки ілюструє, як система реагує на зовнішні події, а також дозволяє проєктувати сценарії взаємодії між користувачем і програмним забезпеченням. Реалізаційна складова UML дозволяє моделювати, як система працюватиме на апаратному забезпеченні, що забезпечує її інтеграцію в реальне середовище.

Окремої уваги заслуговує використання UML для моделювання взаємодії користувачів із системою, де ключову роль відіграють діаграми прецедентів. Ці діаграми описують, як користувачі, представлені в системі у вигляді акторів, взаємодіють із різними аспектами програмного забезпечення.

UML забезпечує гнучкість та інтеграцію в процес розробки, дозволяючи різним спеціалістам працювати разом над створенням ефективних програмних рішень. Розробники можуть використовувати UML для опису технічної архітектури, бізнес-аналітики — для документування бізнес-вимог, а інженери — для проєктування складних технічних систем. Така універсальність забезпечує не лише високий рівень взаєморозуміння між членами команди, а й дозволяє уникати типових помилок, що виникають через нечітке уявлення про функціонування системи.

Важливо зазначити, що використання UML допомагає зменшити ризики, пов'язані з розробкою програмного забезпечення. Чітке графічне подання системи дозволяє виявити проблемні місця ще до початку написання коду, що значно скорочує витрати часу і ресурсів на їхнє усунення в майбутньому. Водночас UML допомагає підвищити якість кінцевого продукту, оскільки завдяки цьому інструменту система проходить багаторівневу верифікацію та аналіз ще на стадії її проєктування.

Основні типи діаграм UML:

- діаграми структури, які описують статичну структуру системи і до яких

відносяться діаграми класів, діаграми компонентів, діаграми розгортання та діаграми пакетів;

- діаграми поведінки [21], які описують поведінку системи і до яких відносяться діаграми станів, діаграми діяльності, діаграми послідовностей та діаграми кооперації;
- діаграми реалізації описують фізичну реалізацію системи, до них відносяться діаграми реалізації класів, діаграми компонентів, діаграми розгортання та діаграми автоматичних машин.

UML є потужним інструментом, який може допомогти розробникам, бізнес-аналітикам і системним інженерам краще зрозуміти і розробити складні системи.

На рисунку 2.8 показано модель Android-системи, для якої було розроблено систему компонентного аналізу з використанням діаграм прецедентів.

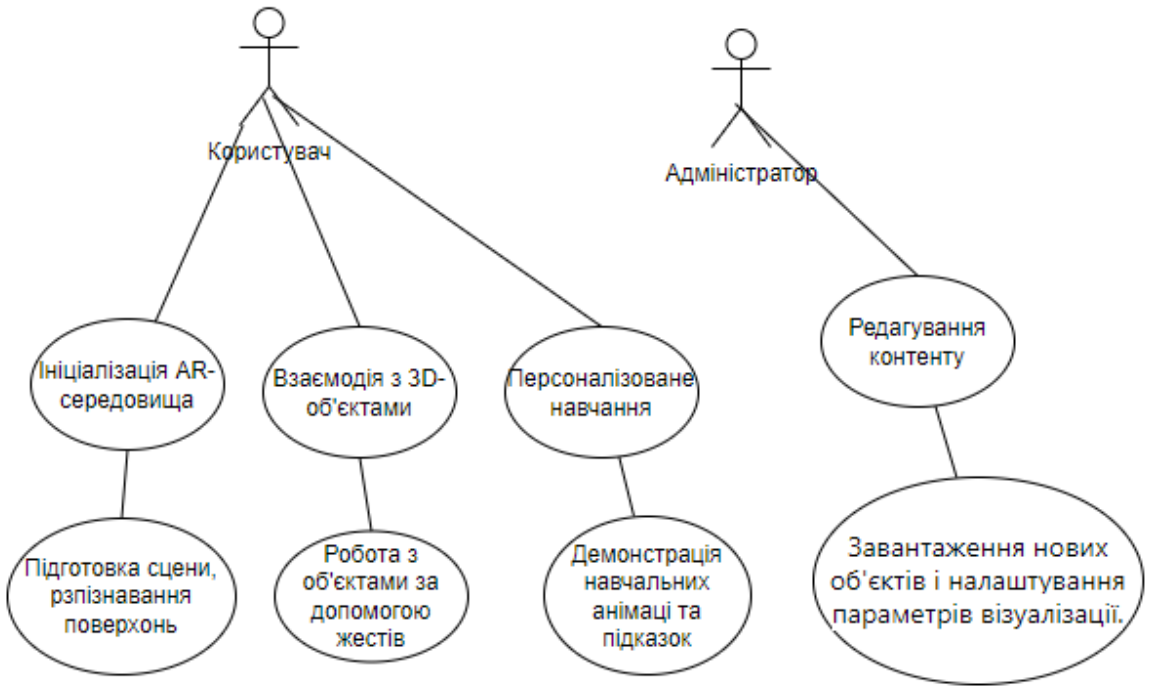

Рисунок 2.8 – Модель Android-системи

Перелік варіантів використання наведено в таблиці 2.1.

Таблиця 2.1 – Реєстр варіантів використання

Актор	Прецедент	Опис
Користувач	Ініціалізація AR-середовища	Підготовка сцени, розпізнавання поверхонь та завантаження моделей.
	Взаємодія з 3D-об'єктами	Робота з об'єктами за допомогою жестів (обертання, масштабування).
	Персоналізоване навчання	Демонстрація навчальних анімацій та підказок.
Адміністратор	Редагування контенту	Завантаження нових об'єктів і налаштування параметрів візуалізації.

Основний функціонал Android-системи AR Room спрямований на спрощення взаємодії користувачів із доповненою реальністю, забезпечуючи доступність цієї високотехнологічної інновації для людей з будь-яким рівнем технічних знань.

2.6 Висновки

У другому У другому розділі було обґрунтовано вибір архітектурного паттерну MVVM, вибір Jetpack Compose для реалізації графічного інтерфейсу, а також розроблено методи, моделі й алгоритми роботи системи, зокрема, для реалізації базового функціоналу системи: персоналізованого навчання користувача, створення 3D об’єктів та виконання операцій над ними, розробка моделі інтерфейсу з можливістю перевикористання у сторонніх сервісах, також створення загальної моделі системи.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ ANDROID-СИСТЕМИ ДЛЯ ВІДОБРАЖЕННЯ ОБ'ЄКТІВ В РЕЖИМІ ДОПОВНЕНОЇ РЕАЛЬНОСТІ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного продукту

Сучасний світ технологій невпинно розвивається, відкриваючи нові горизонти у сфері розробки програмного забезпечення. З кожним днем інжинери спостерігають за появою нових підходів, інструментів та мов програмування, які надають розробникам можливість створювати продукти, що відповідають найвищим стандартам індустрії. Технології розвиваються настільки стрімко, що часто компанії змушені адаптуватися до нових умов ще до того, як встигнуть завершити проєкти з використанням попередніх рішень. У цьому контексті розширення кількості мов програмування не є винятком, а навпаки, стає невід'ємною частиною прогресу.

Вибір мови програмування стає одним із найбільш важливих питань, що постає перед розробниками на початковому етапі створення програмного забезпечення. Це рішення визначає не тільки функціональність і продуктивність майбутнього продукту, але й впливає на всі етапи розробки — від зручності написання коду до можливостей масштабування й підтримки. У світі мобільних додатків, орієнтованих на платформу Android, особливе значення мають мови, які дозволяють створювати зручні графічні інтерфейси, ефективно обробляти дії користувачів і забезпечувати легку роботу з файлами та зовнішніми ресурсами.

Серед найпоширеніших мов, що використовуються для розробки Android-додатків, виділяють Java та Kotlin. Обидві мови мають свої унікальні переваги, що робить їх популярними серед розробників.
Java — це одна з найстаріших і найпоширеніших мов програмування, яка залишається актуальною навіть через десятиліття після свого створення. Заснована на об'єктно-орієнтованій парадигмі, Java стала основою для безлічі програмних рішень. Її синтаксис багато в чому нагадує C і C++, що спрощує

навчання для розробників із досвідом роботи з цими мовами. Однак Java має свою особливість — це універсальність, яка реалізується через принцип "Write Once, Run Anywhere" (Написано раз, виконується всюди). Завдяки цій концепції програми, створені на Java, можуть працювати на будь-яких пристроях, які підтримують віртуальну машину Java (JVM).

Однією з ключових переваг Java є її переносимість. Програми, написані на цій мові, можуть бути легко адаптовані для роботи на різних пристроях і платформах. Це особливо важливо для мобільної розробки, де різноманітність пристроїв створює додаткові виклики. Крім того, Java має багатий набір бібліотек і фреймворків, які значно полегшують розробку складних додатків. Вона підтримує такі важливі функції, як багатопоточність, безпека даних і інтеграція з різними сервісами.

Стабільність і довговічність Java підтверджуються її популярністю. Незважаючи на появу нових мов, таких як Python, Ruby, PHP, Swift, C++ та інших, Java продовжує залишатися одним із лідерів у світі програмування. Це зумовлено її універсальністю, багатофункціональністю та активною підтримкою розробників у всьому світі. До того ж Java часто використовується у великих корпоративних проєктах, де стабільність і масштабованість є ключовими факторами.

Для роботи з Java розробникам необхідно використовувати Java Development Kit (JDK), який включає всі необхідні інструменти для створення, компіляції та виконання програм. Після компіляції програма перетворюється на байт-код, який може бути виконаний у будь-якому середовищі, що підтримує JVM. Це робить Java незамінним інструментом для розробки програмного забезпечення, яке повинно працювати в різних умовах.

Kotlin — це відносно нова мова програмування, створена компанією JetBrains у 2011 році. Вона була розроблена для усунення недоліків Java і надання розробникам сучаснішого інструменту для роботи. Незважаючи на те, що Kotlin спочатку був задуманий як додатковий інструмент для JVM, він швидко здобув популярність серед розробників Android-додатків.

Kotlin має низку переваг, які зробили його вибором номер один для багатьох розробників. Насамперед, це його лаконічність. Код на Kotlin зазвичай значно коротший і зрозуміліший, ніж аналогічний код на Java. Це дозволяє зменшити кількість шаблонного коду і прискорити процес розробки. Крім того, Kotlin забезпечує більш сучасний підхід до програмування, включаючи підтримку асинхронності, що є важливим для роботи з мережею та базами даних.

Окремо слід зазначити особливості Kotlin у роботі з нульовими значеннями. На відміну від Java, де NullPointerException є поширеною проблемою, Kotlin надає механізми для запобігання таких помилок. Це підвищує стабільність додатків і знижує ризик збоїв.

У 2019 році Google офіційно оголосила Kotlin основною мовою для розробки додатків під Android. Це рішення підтвердило переваги Kotlin і стимулювало його подальше поширення серед спільноти розробників. Сьогодні Kotlin використовується не тільки для створення Android-додатків, але й для серверних рішень, веброзробки та навіть розробки ігор.

Популярність Kotlin пояснюється ще й тим, що він легко інтегрується з Java. Це дозволяє командам, які працювали на Java, поступово переходити на Kotlin без значних витрат часу і ресурсів.

Вибір між Java та Kotlin залежить від специфіки проєкту, вимог замовника і навичок команди. Java забезпечує стабільність, перевірену часом, і широку екосистему, тоді як Kotlin пропонує сучасний підхід, зручність і більшу продуктивність. Обидві мови залишаються важливими інструментами для розробників, які працюють у світі Android. Оскільки Android займає понад 70% ринку мобільних пристроїв, знання цих мов відкриває величезні можливості для розробників у створенні інноваційного та високоякісного програмного забезпечення.

Результати порівняння розглянутих мов програмування за обраними критеріями наведені у таблиці 3.1.

Таблиця 3.1 – Порівняльні характеристики мобільних додатків

Мова програмування / Критерій	Java	Kotlin
Функціональні розширення	-	+
Розробка під	+	+
Підтримка ООП	+	+
Sealed classes	-	+
Null safety	-	+
Підсумковий результат	2	5

Таким чином, можна зробити висновок, що Kotlin є однією з кращих мов програмування серед порівнюваних мов і відповідає всім вимогам для розробки Android-системи «AR Room».

3.2 Розробка програмного модуля персоналізації навчання користувача

Модуль персоналізації навчання в додатку є ключовим елементом, що спрямований на створення індивідуального досвіду користувача. У нашій системі розробка такого модуля базується на інтерактивному вивченні жестів за допомогою доповненої реальності, використовуючи механізми анімацій та візуального супроводу.

Персоналізація передбачає адаптацію навчального процесу під кожного окремого користувача, враховуючи його рівень підготовки та індивідуальні потреби. Основними функціями цього модуля є:

1. Інтерактивна демонстрація ключових жестів, що необхідні для використання функціоналу програми;

2. Вивчення проходить поетапно, починаючи з базових рухів і завершуючи складними комбінаціями;
3. Навчання супроводжується анімаціями, що відображають правильність виконання рухів, а також текстовими підказками;
4. Користувач має змогу самостійно обирати кількість повторень та швидкість навчання.

Модуль реалізовано за допомогою спеціальних компонентів на базі Jetpack Compose. Для цього використано механізм анімацій Lottie [22], що дозволяє відтворювати інтерактивні візуалізації з високою продуктивністю. Нижче наведено основні аспекти реалізації (рис. 3.1).

```
@Composable
private fun LottieView(
    modifier: Modifier = Modifier,
    compositionSpec: LottieCompositionSpec,
    onEnd: () -> Unit
) {
    val composition by rememberLottieComposition(compositionSpec)
    val progress by animateLottieCompositionAsState(composition)
    LottieAnimation(
        modifier = modifier.aspectRatio(LocalDimens.ARView.GestureAnimRatio),
        composition = composition,
        contentScale = ContentScale.FillWidth,
        progress = {
            if (progress == Constants.ANIM_COMPLETE_VALUE) {
                onEnd()
            }
            progress
        }
    )
}
```

Рисунок 3.1 – Функція LottieView

Основу модуля складають такі методи:

1. Метод GestureOverlay відповідає за створення послідовності анімацій для кожного з жестів, які необхідно вивчити.

2. Компонент LottieView відповідає за відтворення анімацій. Він дозволяє відстежувати прогрес і автоматично переходити до наступного жесту після завершення анімації.
3. Для спрощення взаємодії реалізовано окремий компонент ARCoachingOverlay, який відображає текстові повідомлення з підказками у процесі навчання. Користувач не просто отримує інформацію, а активно взаємодіє з додатком, виконуючи вказівки на екрані. Можливість адаптації тривалості навчання та кількості повторень забезпечує комфортне освоєння нового матеріалу. Використання Jetpack Compose і Lottie гарантує плавність анімацій навіть на пристроях з обмеженими ресурсами.

   Все вищеописане є складовою класу ARSceneView.

   Діаграма класів подана на рисунку 3.2.

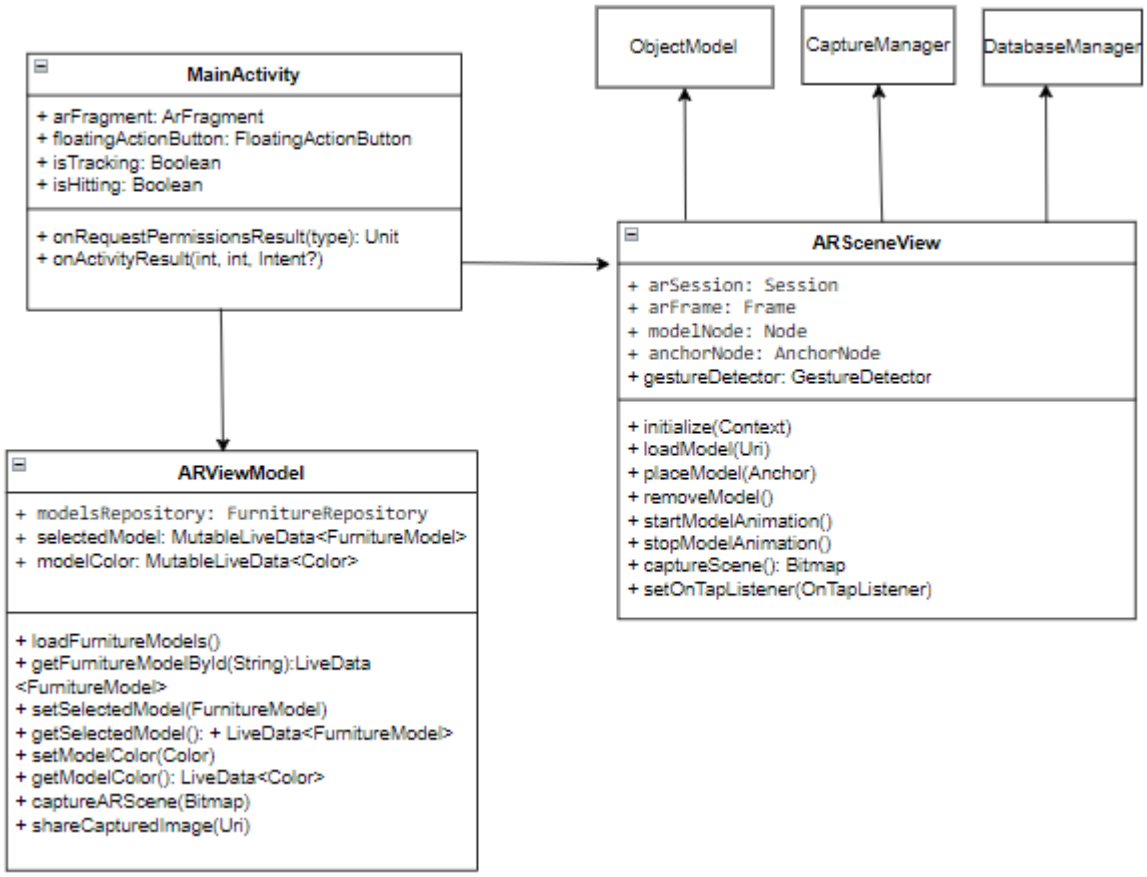

Рисунок 3.2 – Діаграма класів

Модуль персоналізації навчання користувачів є важливою частиною додатка, спрямованою на покращення користувацького досвіду. Реалізація такого модуля за допомогою сучасних технологій забезпечує інтерактивність, гнучкість і високу ефективність навчання. Це дозволяє користувачам швидко освоїти функціонал додатка і комфортно використовувати його можливості.

3.3 Розробка програмного модуля фільтрації об’єктів

Розробка програмного модуля фільтрації об’єктів є невід'ємною складовою будь-якого сучасного програмного забезпечення, що передбачає роботу з великим масивом даних. Модуль фільтрації забезпечує зручний та інтуїтивно зрозумілий інтерфейс для користувачів, дозволяючи їм знаходити потрібні об'єкти відповідно до заданих критеріїв, значно підвищуючи ефективність роботи з додатком. Код, що наведено нижче, описує реалізацію такого модуля за допомогою декларативного підходу, використовуючи Jetpack Compose — сучасний інструмент для створення інтерфейсів у середовищі Android.

Основні функції модуля:

1) користувач може швидко вибрати потрібну категорію зі списку доступних, щоб зосередитися лише на цікавих йому об’єктах;

2) зміна вибраної категорії одразу викликає оновлення даних, які відображаються на екрані;

3) використання горизонтального списку з можливістю прокручування забезпечує компактне розташування категорій і доступність для користувача;

4) компоненти модуля реалізовано у вигляді кнопок-фільтрів , які інтуїтивно зрозумілі та легкі у використанні.

Основою модуля є функція CategoryHeader, яка відповідає за відображення категорій у вигляді горизонтального списку. Для цього застосовується компонент LazyRow, який оптимізовано для роботи з великими списками. Завдяки властивості LazyRow, категорії динамічно створюються та знищуються

у пам'яті залежно від їхньої видимості на екрані, що забезпечує високу продуктивність навіть у випадках, коли кількість категорій є значною.

Елементи категорій у цьому списку створюються за допомогою функції items, яка перебирає список категорій і викликає для кожної з них функцію CategoryCard. Важливою частиною є налаштування відступів та відступу вмісту за допомогою PaddingValues і властивостей Arrangement.spacedBy. Це дозволяє досягти естетичного та рівномірного розташування категорій на екрані, зберігаючи при цьому гармонійний дизайн.

Функція CategoryCard відповідає за відображення кожного окремого елемента категорії. Вона реалізована через компонент FilterChip, який надає користувачам можливість вибору потрібної категорії. Особливістю цієї реалізації є використання параметра isSelected, який визначає, чи є поточна категорія вибраною. Це дозволяє візуально виділити активну категорію, полегшуючи навігацію для користувача.

Ще однією ключовою особливістю реалізації є інтеграція з багатомовною підтримкою додатка. Текст для відображення назви категорії отримується через функцію stringResource, яка автоматично підбирає локалізований рядок залежно від вибраної мови додатка. Це значно підвищує доступність додатка для користувачів з різних регіонів світу.

Механізм зміни вибраної категорії реалізований за допомогою замикання onClick, яке викликається при натисканні на категорію. В параметр onCategoryChange передається нова вибрана категорія, що дозволяє програмі оновити стан модуля та відобразити відфільтровані результати. Такий підхід забезпечує гнучкість та масштабованість, дозволяючи без значних змін у коді додавати нові категорії або змінювати їхню структуру.

Переваги реалізації:

1) використання LazyRow дозволяє оптимально працювати навіть із великими списками категорій, забезпечуючи плавне прокручування та мінімальне навантаження на систему;

2) компонент FilterChip легко адаптується до різних стилів і тем додатка, що робить модуль сумісним із будь-яким дизайном;
3) простий і зрозумілий інтерфейс сприяє легкому освоєнню модуля користувачами без попереднього навчання;
4) розділення логіки на окремі компоненти CategoryHeader і CategoryCard спрощує підтримку та розширення функціональності модуля.

Алгоритм роботи модуля подано на рисунку 3.3

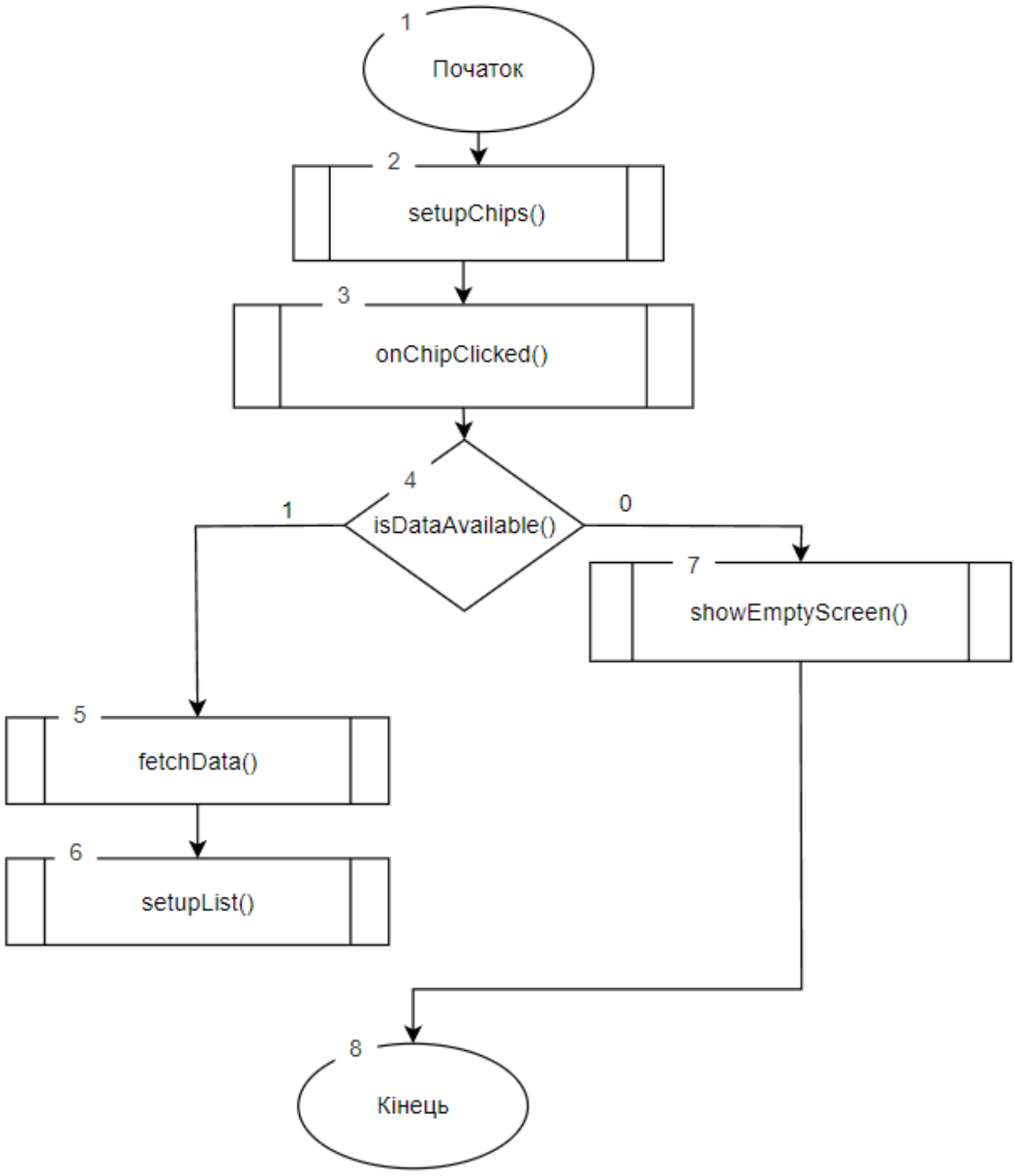

Рисунок 3.3 – Алгоритм роботи модуля фільтрацій

У підсумку, цей модуль надає користувачам ефективний спосіб взаємодії з великими наборами даних. Завдяки використанню компонентів Jetpack Compose та декларативного підходу до розробки, вдалося досягти елегантного та продуктивного рішення, яке забезпечує високу інтерактивність та зручність у використанні. Застосування таких сучасних підходів дозволяє розробникам створювати інтерфейси, які не лише задовольняють функціональні вимоги, а й відповідають високим стандартам користувацького досвіду.

3.4 Розробка програмного модуля відображення об’єктів

Модуль відображення об'єктів у доповненій реальності (AR) є центральною частиною додатку, що дозволяє користувачам інтерактивно взаємодіяти з віртуальними елементами, інтегрованими у фізичний простір. Завдяки цьому модулю віртуальні моделі, такі як 3D-об'єкти або текстури, можуть бути відображені у реальному світі з урахуванням його контексту та властивостей. У цьому розділі детально розглянуто код реалізації функції ARScene, яка відповідає за створення сцени з об'єктами, що відображаються в AR, її можливості та особливості.

Модуль реалізує компонент ARScene, який є основою для візуалізації об’єктів у доповненій реальності. Він включає такі базові елементи:

Візуалізація сцени була зроблена за допомогою компоненту ARScene, що займає весь екран, що забезпечується використанням модифікатора modifier.fillMaxSize(). Це дозволяє ефективно використовувати простір для відображення AR-елементів. CameraNode відповідає за розташування та орієнтацію "віртуальної камери", що визначає точку огляду у сцені. childNodes — список дочірніх вузлів, які представляють додані об'єкти.

Інтерактивність за допомогою жестів додана завдяки rememberOnGestureListener (рис. 3.4), що обробляє основні жести користувача:

- Одиночне натискання (onSingleTapConfirmed) — використовується для додавання моделі у сцену, запуску її анімації та активації інформаційного інтерфейсу.

- Переміщення (onMove) — дозволяє змінювати розташування об'єкта на сцені шляхом перетягування. Для цього змінюється позиція "батьківського" вузла (AnchorNode), який пов’язаний з моделлю.

Інтерактивні елементи створюють реалістичний досвід для користувача, дозволяючи йому безпосередньо маніпулювати об’єктами на екрані.

Налаштування сесії AR було зроблене за допомогою Функціональнісі sessionCameraConfig, що дозволяє вимикати непотрібні елементи, такі як конфігурація глибини (session.disableDepthConfig()), щоб оптимізувати продуктивність. SessionConfiguration динамічно змінює параметри конфігурації сесії відповідно до поточних потреб, наприклад, змінюючи режим знаходження площин (planeFindingMode).

```
onGestureListener = rememberOnGestureListener(
    onSingleTapConfirmed = { _, _ ->
        if (animateModel) {
            onModelPlace()
            animateModel = false
            showGestureOverlay = true
        }
        onStopRotation()
    },
    onMove = { _, event, node ->
        if (node == null) return@rememberOnGestureListener
        // Move gesture to move model node. Instead of changing model position
        // change anchor node position (parent of model node)
        frame?.hitTest(event)?.firstOrNull()?.hitPose?.position?.let { position ->
            val anchor = (node.parent as? AnchorNode) ?: node
            anchor.worldPosition = position
        }
    }
),
```

Рисунок 3.4 – Додавання інтерактивності за допомогою жестів

Оновлення стану відбувається за допомогою onTrackingFailureChanged, активується "коучинг", який відображає підказки для допомоги користувачеві у

відновленні роботи AR. Функція onSessionUpdated відслідковує зміни стану кадру та оновлює положення чи властивості об'єктів на сцені в реальному часі. Наприклад, це може бути зміна позиції моделі або динамічне оновлення кольору. Якщо сцена порожня (тобто childNodes не містить елементів), функція onSessionUpdated створює новий "якірний" вузол (AnchorNode). Цей вузол слугує точкою прив'язки для моделі, яку додано до сцени. Процес виглядає так:

1) використання функції createCenterAnchorNode для визначення центра сцени;
2) додавання об’єкта моделі (model) до створеного вузла;
3) збереження вузла у списку childNodes.

Коли параметр animateModel активований, об’єкт, розташований у сцені, автоматично анімується. Анімація синхронізується з положенням користувача, що створює ефект живого взаємодії з моделлю (рис. 3.5).

```
if (childNodes.isEmpty()) {
    updatedFrame.createCenterAnchorNode(engine)?.let { anchorNode ->
        model?.let {
            animateModel = true
            anchorNode.addChildNode(it)
            childNodes.add(anchorNode)
            planeRenderer = false
            showCoachingOverlay = false
        }
    }
} else if (animateModel) {
    (childNodes.firstOrNull() as? AnchorNode)?.apply {
        val (x, y) = view.viewport.run {
            with(LocalDimens.ARView) {
                width / MODEL_PLACEMENT_WIDTH_PROPORTION to height / MODEL_PLACEMENT_HEIGHT
            }
        }

        val newPosition = updatedFrame.getPose(x, y)?.position ?: return@ARScene
        worldPosition = Position(newPosition.x, newPosition.y, newPosition.z)
    }
}
```

Рисунок 3.5 – робота з під-елементами та анімованою моделлю

Функціонал onMove дозволяє змінювати позицію об'єктів, реагуючи на жести переміщення. Замість прямого зміщення моделі змінюється положення її "якірного" вузла. Це зменшує складність і забезпечує коректну інтеграцію моделі у сцену.

Якщо активовано параметр isDynamicColor, об’єкт у сцені змінює свій колір на основі поточного освітлення чи інших умов кадру. Це відбувається завдяки функції getDynamicColor, яка обчислює новий колір кожні кілька мілісекунд.

Модуль відображення об’єктів у AR підтримує низку функцій, спрямованих на оптимізацію роботи:

- конфігурації сесії та стану сцени можуть динамічно змінюватися, що дозволяє адаптувати функціональність до конкретних завдань або вимог;
- використання асинхронних функцій дозволяє мінімізувати затримки при оновленні кадру чи обробці жестів.

З переваг створеного підходу можна виділити:

1) користувачі можуть легко взаємодіяти з об'єктами за допомогою жестів, таких як переміщення чи натискання;
2) об'єкти інтегруються у фізичний простір з урахуванням його властивостей;
3) розробники мають змогу адаптувати параметри модуля до конкретних сценаріїв використання.

Алгоритм роботи модуля складається з наступних кроків:

1. ARSceneView створює та налаштовує AR-сесію, використовуючи ARCore для відстеження оточення та позиції пристрою.
2. ModelLoader завантажує 3D-модель меблів з локального сховища або через API SketchFab, використовуючи наданий шлях або ідентифікатор моделі.
3. GestureDetector відстежує жести користувача, такі як торкання, переміщення, масштабування та обертання, для взаємодії з 3D-моделлю.
4. AnchorNode створює якір у визначеній точці простору, де буде розміщена модель.

5. ModelNode прив'язує 3D-модель до створеного якоря для стабільного відображення в реальному середовищі.
6. MaterialInstance дозволяє змінювати колір та інші матеріальні властивості моделі відповідно до вибору користувача.
7. AnimationController керує анімаціями моделі, такими як обертання або інші ефекти.
8. ARSceneView рендерить 3D-модель у реальному часі, інтегруючи її в оточення користувача за допомогою камери пристрою.
9. CoachingOverlay відображає підказки та інструкції для покращення взаємодії користувача з AR-сценою.

Розроблений модуль є основою для створення AR-додатку. Завдяки функціям динамічного додавання об’єктів, інтерактивності, а також адаптивному налаштуванню, цей модуль дозволяє створювати високоякісні рішення з багатим функціоналом та зручним користувацьким досвідом.

3.5 Розробка програмного модуля завантаження моделей для відображення

Модуль завантаження моделей для відображення є важливим компонентом, що забезпечує динамічну інтеграцію 3D-моделей у додаток. Він надає функціонал для отримання інформації про модель, завантаження файлів з віддаленого сервера та збереження їх локально. Це дозволяє зберігати моделі для подальшого використання без необхідності повторного завантаження, що оптимізує швидкість роботи програми та забезпечує кращий користувацький досвід.

Функція download(productId: String) виконує завдання завантаження моделі, починаючи з отримання інформації про її розташування та розмір. Для цього використовується getDownloadInfoUseCase, який повертає потокові дані про стан завантаження у вигляді об'єкта Result:

- Result.Loading — відображає стан завантаження інформації про модель;

- Result.Success — Містить дані про URL для завантаження моделі та її розмір;
- Result.Error — Вказує на помилки при отриманні інформації про модель.

Перед початком завантаження перевіряється, чи модель уже завантажена локально, використовуючи метод checkIfExist. Якщо файл існує і його розмір збігається із заявленим у метаданих, завантаження не виконується, а модуль переходить до наступного етапу.

Якщо модель не знайдено або файл неповний, викликається функція startDownload, яка виконує завантаження моделі з URL, вказаного у відповідних метаданих. Завантаження виконується асинхронно у межах viewModelScope, що дозволяє уникнути блокування основного потоку програми.

Під час завантаження функція startDownload реагує на різні стани через об'єкт DownloadStatus:

- DownloadStatus.Loading — вказує на те, що завантаження активне, але дані ще не доступні;
- DownloadStatus.Progress — відображає прогрес завантаження, зокрема, кількість завантажених байтів;
- DownloadStatus.Completed — сигналізує про успішне завершення завантаження файлу;
- DownloadStatus.Error — виникає у випадку, якщо завантаження не вдалося через помилку, наприклад, через втрату з’єднання.

Алгоритм роботи модуля (рис. 3.6):

1) при виклику download(productId) отримується локальний шлях для зберігання моделі;
2) запускається асинхронний процес для отримання метаданих моделі getDownloadInfoUseCase;
3) якщо модель уже завантажено і розмір файлу відповідає метаданим, вона вважається завантаженою, і додаток переходить до етапу використання моделі;

4) у разі відсутності файлу або невідповідності його розміру починається завантаження через startDownload, яке враховує прогрес та можливі помилки;
5) якщо завантаження не вдалося, модуль перевіряє, чи існує кешована версія файлу, і за можливості використовує її.

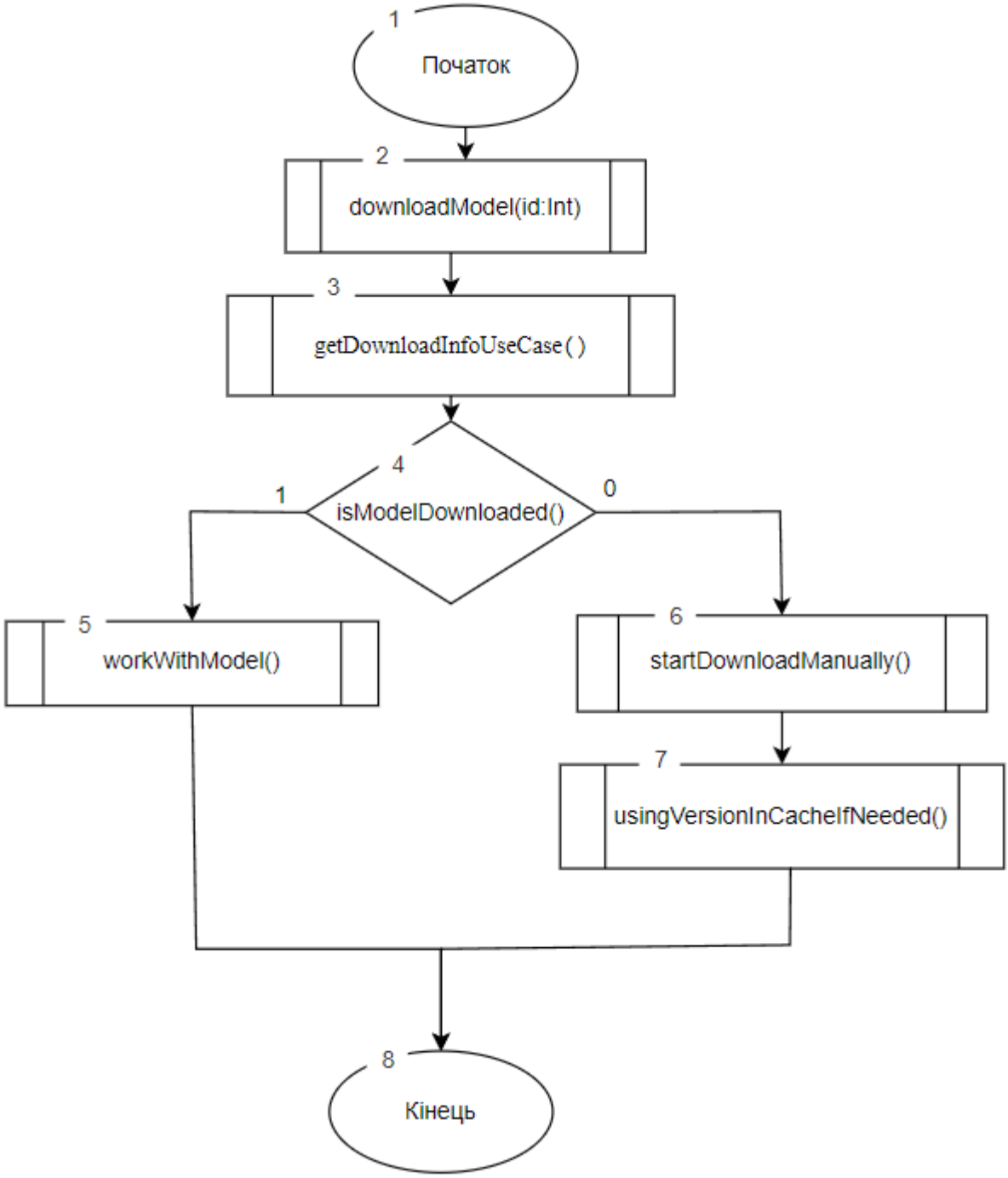

Рисунок 3.6 – Алгоритм роботи модуля завантажень

З особливостей реалізації виділено:

1) завдяки використанню viewModelScope і корутин, процес завантаження виконується без блокування основного потоку, що забезпечує плавність роботи додатка;
2) стан завантаження передається у вигляді об’єкта downloadUiState, що дозволяє відображати відповідні повідомлення або індикатори прогресу для користувача;
3) завдяки поділу завантаження на етапи, модуль легко адаптується для завантаження моделей різних форматів або обробки специфічних помилок;
4) використання локального збереження моделей оптимізує роботу програми при повторному доступі до тих самих даних.

Переваги модуля:

1) перевірка наявності файлу перед завантаженням дозволяє уникнути зайвих запитів до сервера;
2) відображення прогресу завантаження забезпечує користувача зрозумілим інтерфейсом.

Модуль стійкий до помилок завдяки підтримці кешованих даних і обробці помилок завантаження.

Розроблений модуль завантаження моделей демонструє високу гнучкість і надійність, забезпечуючи ефективну взаємодію між серверною і клієнтською частинами додатка. Завдяки реалізації кешування, асинхронності та обробки помилок цей модуль відповідає вимогам сучасних AR-додатків, де швидкість і стабільність роботи відіграють ключову роль.

3.6 Розробка програмного модуля для модифікації 3D об’єктів в ході роботи застосунку

Розробка модуля для модифікації 3D об'єктів у реальному часі є важливою складовою сучасних застосунків, які працюють із візуалізацією, такими як програми для дизайну, доповненої реальності чи інтерактивних презентацій. Цей

модуль забезпечує можливість гнучко змінювати параметри моделей, зокрема їхній колір, дозволяючи користувачам адаптувати вигляд об'єктів під свої потреби або контекст використання. Однією з ключових функцій, що надаються, є інтерактивний вибір кольору з палітри. Ця функція реалізується через спеціальний компонент, який включає в себе як інструменти для вибору статичних кольорів, так і можливість вмикати динамічну зміну кольору. Також передбачена функція скидання до початкового кольору, яка допомагає швидко повернутися до базового стану об’єкта.

Основні можливості модуля

1) користувач може обирати колір моделі за допомогою кольорового палітрового вибірника;
2) передбачено можливість використання динамічних кольорів, які змінюються в залежності від зовнішніх факторів або даних;
3) є функція скидання кольору до оригінального стану, що забезпечує збереження вихідного вигляду моделі;
4) компоненти інтерфейсу, такі як кнопки, забезпечують зручний доступ до функцій модуля;
5) використовується адаптивний підхід для коректного відображення і взаємодії, незалежно від розмірів екрана;
6) для покращення користувацького досвіду передбачено анімовані переходи між станами, наприклад, при відкритті чи закритті палітри кольорів.

Уся взаємодія з модулем відбувається через зручний та інтуїтивно зрозумілий інтерфейс. Наприклад, елементи керування розташовані у вигляді рядка кнопок, де кожна кнопка відповідає за певну функцію: вибір кольору, активацію динамічного режиму чи відкриття палітри. У компонентах використовується адаптивний дизайн, що дозволяє коректно відображати елементи на пристроях із різними розмірами екрана, забезпечуючи зручний доступ до функцій. Важливим аспектом є те, що модуль включає анімацію для покращення вражень від використання. Наприклад, при відкритті вибірника

кольорів спостерігається плавне розширення області з палітрою, що створює відчуття природності та інтерактивності.

Механізм зміни кольору базується на реактивному підході, коли стан модуля оновлюється залежно від дій користувача. Наприклад, коли обирається новий колір, система автоматично оновлює матеріали моделі, забезпечуючи миттєвий візуальний результат. Реактивність досягається за допомогою LaunchedEffect, який дозволяє реагувати на зміну стану кольору та змінювати параметри моделі відповідно до поточних налаштувань (рис. 3.7). У разі активації динамічного режиму кольору система може постійно змінювати вигляд об’єкта залежно від зовнішніх умов або внутрішньої логіки. Наприклад, якщо користувач обирає динамічний режим, кольори можуть змінюватися в режимі реального часу, створюючи ефект "живої" моделі. Це додає глибини та інтерактивності до взаємодії з програмою.

Важливою частиною розробки цього модуля є реалізація механізму збереження оригінального вигляду моделі. Це дає змогу зберігати початковий стан об'єкта, щоб користувач міг у будь-який момент повернути модель до її первинного вигляду. Наприклад, якщо об'єкт має базові матеріали, вони зберігаються в окремій змінній, що дозволяє їх легко відновити.

Крім того, для створення більш привабливого досвіду використання, розробники інтегрували систему анімацій. При зміні стану кольорового вибірника або активації додаткових функцій, таких як динамічний режим, елементи інтерфейсу плавно змінюють розміри або додають нові елементи. Це робить взаємодію більш природною та зрозумілою навіть для користувачів, які вперше працюють із програмою. Такий підхід дозволяє покращити загальний досвід використання та підвищити зацікавленість.

```
LaunchedEffect(key1 = selectedColor, key2 = model) {
    when (selectedColor) {
        is ColorState.Color -> {
            colorMaterials?.let { model?.materialInstances = it }
            currentColor = selectedColor.value
        }

        ColorState.Dynamic -> {
            colorMaterials?.let { model?.materialInstances = it }
        }

        ColorState.None -> {
            originalMaterials?.let { model?.materialInstances = it }
            currentColor = Color.White
        }
    }
}
```

Рисунок 3.7 – Зміна характеристик моделі у відповідь на зміну кольору

Кодова реалізація включає модуль ColorOption, що є центром взаємодії з кольоровими налаштуваннями. Вона поєднує в собі логіку роботи з кольоровими станами, інтерактивність і реактивний підхід до управління параметрами моделі. Наприклад, функція для вибору кольору динамічно змінює параметри матеріалів моделі залежно від обраного користувачем значення, а також дозволяє швидко перемикатися між різними режимами кольору за допомогою відповідних кнопок. Важливо, що модуль враховує збереження стану між перезавантаженнями, дозволяючи продовжувати роботу з моделлю без втрати налаштувань.

Аглоритм роботи модуля застосунку (рис. 3.8):

1. Модель 3D-об'єкта завантажується через модуль завантаження і додається до AR-сцени як ModelNode.Встановлюється початковий масштаб, положення, обертання, та матеріали.

2. Змінюється позиція моделі у просторі, оновлюючи її worldPosition. Змінюється масштаб моделі через оновлення її scale.Оновлюється rotation моделі відповідно до жесту обертання.
3. MaterialInstance використовується для зміни матеріалів моделі. Новий колір задається через виклик setColor(color: Color).Для текстур застосовується метод setTexture(textureUri: Uri).
4. AnimationController активує або зупиняє анімації. ItartAnimation(name: String) — запускає анімацію моделі, stopAnimation() — зупиняє поточну анімацію. За потреби додаються циклічні або одноразові анімації.
5. Модель оновлюється залежно від зовнішніх факторів. Отримуються дані від камери або сенсорів. Модель оновлюється за допомогою викликів updateColor(detectedColor: Color).
6. Після кожної модифікації модель автоматично перерисовується через ARSceneView.
7. Зміни синхронізуються з об’єктом ARViewModel для відображення актуального стану в інтерфейсі користувача.
8. Реалізується можливість повернення до початкового стану. Масштаб, обертання та матеріали скидаються до значень, збережених під час завантаження.

Розробка цього модуля є прикладом інтеграції сучасних технологій для створення потужного та зручного інструмента для роботи з 3D об’єктами. Завдяки широкому функціоналу та увазі до деталей, цей модуль дозволяє користувачам легко адаптувати зовнішній вигляд моделей до своїх потреб і робити це з максимальним комфортом.

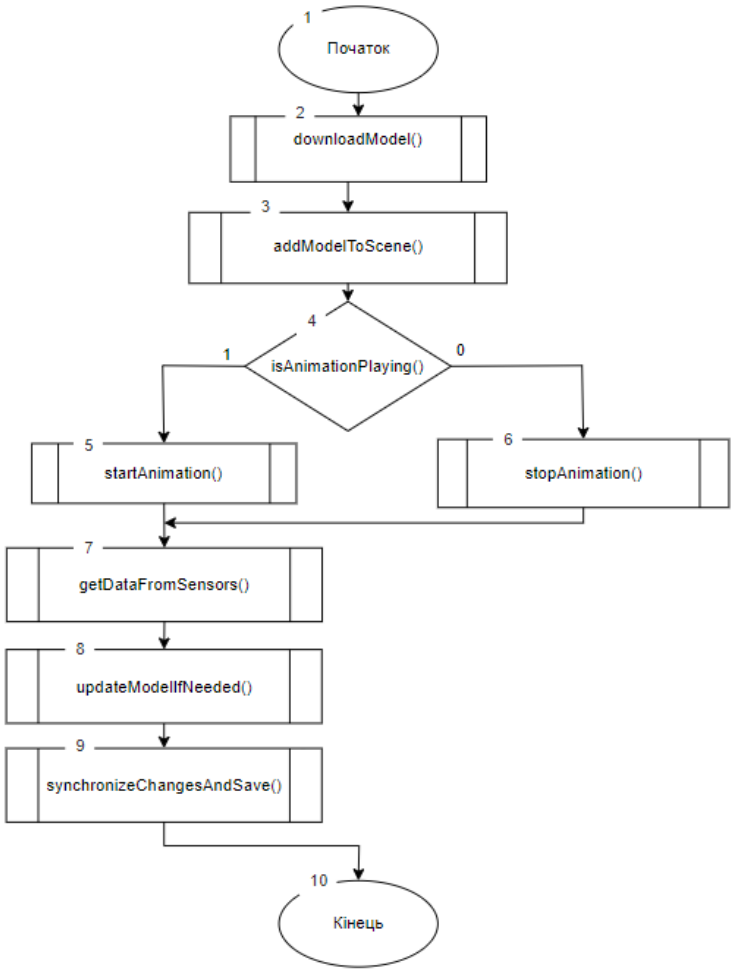

Рисунок 3.8 – Алгоритм модуля для модифікації 3D об'єктів

3.7 Висновки

У третьому розділі було здійснено обґрунтований вибір засобів програмування та технологій, які використано при розробці програмного продукту, обрано середовище розробки, а також розроблено усі модулі програми.

Було обрано мову програмування Kotlin, інтегроване середовище розробки Android Studio і для написання графічного інтерфейсу користувача – Jetpack Compose.

Розроблено Android-систему «AR Room», яка включає модуль персоналізації навчання користувача; модуль модуля фільтрації об’єктів; модуль відображення об’єктів; модуль завантаження моделей для відображення; модуль для модифікації 3D об’єктів в ході роботи застосунку.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Аналіз методів тестування

Тестування програмного забезпечення є ключовим етапом у розробці, який передбачає систематичне проведення експериментів з метою виявлення помилок і дефектів у програмі. Цей процес дозволяє переконатися, що програма відповідає вимогам користувачів, працює стабільно, надійно та безпечно, а також задовольняє всі функціональні та нефункціональні вимоги. Тестування є не лише перевіркою коректності роботи, але й важливим інструментом забезпечення якості програмного продукту.

На початкових етапах розвитку програмних систем, коли вони створювалися переважно для задоволення потреб наукових досліджень або військових замовлень, тестування мало суворо формалізований характер. Всі процедури виконувалися за заздалегідь розробленими інструкціями, а результати ретельно документувалися. На той час тестування зазвичай починалося лише після завершення процесу кодування, і в багатьох випадках його виконували ті самі люди, які брали участь у розробці. Це створювало певні труднощі, оскільки упередженість розробників могла впливати на результати тестування.

Традиційно основний акцент робився на так званому "вичерпному" тестуванні, коли програма перевірялася з використанням всіх можливих шляхів у коді або максимальної кількості варіантів вхідних даних. Проте згодом стало зрозуміло, що цей підхід має суттєві обмеження. Повне охоплення всіх можливих сценаріїв є практично неможливим через величезну кількість комбінацій вхідних даних, шляхів виконання та логічних умов. Крім того, цей підхід не дозволяв виявляти проблеми, пов'язані з архітектурними недоліками або помилками в специфікаціях.

Згодом тестування почали розглядати як процес, спрямований на демонстрацію правильності роботи програмного забезпечення. Однак цей підхід був поступово замінений більш сучасним поглядом, за яким головною метою

тестування є пошук помилок і недоліків у роботі програми. Таким чином, успішне тестування – це те, яке виявляє раніше невідомі дефекти, а не лише підтверджує коректність роботи програмного забезпечення. Цей підхід часто називають "парадоксом тестування", оскільки він одночасно ставить за мету як підтвердження якості продукту, так і виявлення його недоліків.

Тестування може бути поділене на два основні види: статичне та динамічне. Статичне тестування передбачає перевірку програмного коду, документації та інших артефактів без фактичного виконання програми. Воно включає аналіз технічного завдання, специфікацій, вихідного коду та іншої документації на предмет відповідності встановленим стандартам. Цей підхід дозволяє виявити помилки та невідповідності на ранніх етапах розробки, що значно знижує витрати на виправлення дефектів у майбутньому.

Динамічне тестування, навпаки, виконується безпосередньо під час роботи програми. Воно включає запуск тестів із заздалегідь підготовленими наборами вхідних даних і аналіз результатів роботи програми. Динамічні методи дозволяють перевірити функціональність програми та виявити помилки, які можуть проявлятися лише під час її виконання.

Серед методів тестування виділяють три основних підходи: "чорну скриньку" [23], "білу скриньку" [24] та "сіру скриньку" [25]. Тестування "чорної скриньки" полягає у перевірці програми без знання її внутрішньої структури. Тестувальник працює так само, як кінцевий користувач, вводячи дані та аналізуючи отримані результати. Цей підхід дозволяє виявляти помилки у функціонуванні програми незалежно від того, як вона реалізована.

Тестування "білої скриньки", навпаки, базується на аналізі внутрішньої структури програми. Тестувальник має доступ до вихідного коду і перевіряє, чи правильно реалізовані всі алгоритми, чи охоплюються всі можливі сценарії виконання, та чи відповідає програма специфікаціям. Цей метод є ефективним для виявлення логічних та структурних помилок.

Метод "сірої скриньки" поєднує в собі елементи двох попередніх підходів. Тестувальник має часткове розуміння внутрішньої структури програми, що

дозволяє йому враховувати як зовнішню поведінку системи, так і особливості її реалізації. Цей метод є ефективним для комплексного аналізу програми, оскільки дозволяє врахувати як перспективу користувача, так і технічні аспекти.

Після аналізу всіх методів було вирішено, що оптимальним підходом для тестування мобільного додатку стане використання методу "чорної скриньки". Цей підхід дозволяє виявити помилки у функціонуванні програми на ранніх етапах розробки, не вимагаючи глибокого розуміння її внутрішньої структури. Такий вибір забезпечує ефективність тестування та дозволяє зосередитися на поліпшенні користувацького досвіду.

4.2 Тестування програмного продукту

Коли користувач відкриває мобільний додаток «AR Room» на Android-пристрої, першим екраном, який він бачить, є Splash screen [26] — звичний елемент інтерфейсу більшості сучасних мобільних додатків (див. рисунок 4.1).

Рисунок 4.1 – Splash screen додатку «AR Room»

На рисунку 4.2 відображено інтерфейс застосунку, який призначений для вибору та перегляду об'єктів меблів. Основна ідея інтерфейсу — надати користувачеві можливість швидко й ефективно знайти необхідні елементи меблів за категоріями та переглянути їх візуальні моделі.

На першому екрані представлена категорія меблів під назвою «Sofa». У верхній частині екрану розташовані вкладки, які дають змогу користувачеві перемикатися між різними категоріями, такими як «Table», «Chair», «Bed», «Sofa», «Desk», і «AC». Активною категорією є «Sofa», що виділяється темнішою підсвіткою.

У центральній частині екрану відображено кілька моделей диванів у вигляді карток, кожна з яких містить візуалізацію та коротку назву. Також представлено вибори інших категорій з відповідним дизайном.

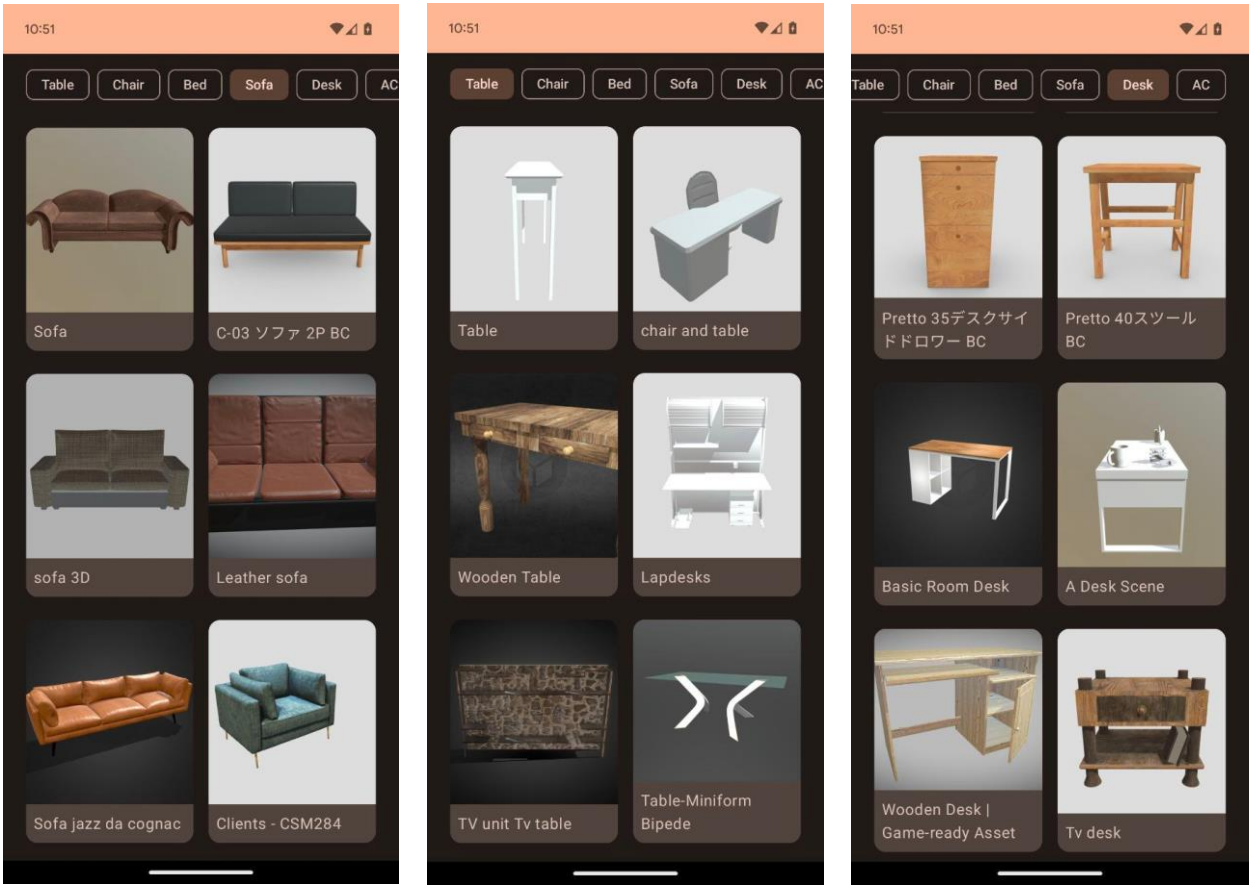

Рисунок 4.2 – Екран з перегляним доступних для використання товарів

Усі екрани мають уніфікований дизайн, який включає темну кольорову гаму та світлі елементи для підкреслення контенту. Картки з моделями меблів розташовані у вигляді сітки, що забезпечує легкість навігації та швидкий перегляд. Користувач може натискати на кожну картку для отримання детальнішої інформації про модель меблів.

Основна мета інтерфейсу — забезпечити простоту використання та доступ до великої кількості моделей меблів. Застосунок орієнтований на користувачів, які шукають меблі для дизайну інтер'єру, покупки або створення 3D-візуалізацій.

На рисунку 4.3 зображено екран, що відображає стан завантаження, який є частиною процесу взаємодії користувача із застосунком. Основна увага зосереджена на центральній частині екрана, де зображено великий значок хмари зі стрілкою вниз. Цей символ є універсальним маркером для позначення процесу завантаження даних з інтернету або іншого зовнішнього джерела. Нижче значка розташовано графічний елемент у вигляді обертового півкола, який виконує функцію індикатора активності. Постійний рух цього індикатора створює у користувача враження, що завантаження триває, навіть якщо немає текстових або числових даних про прогрес.

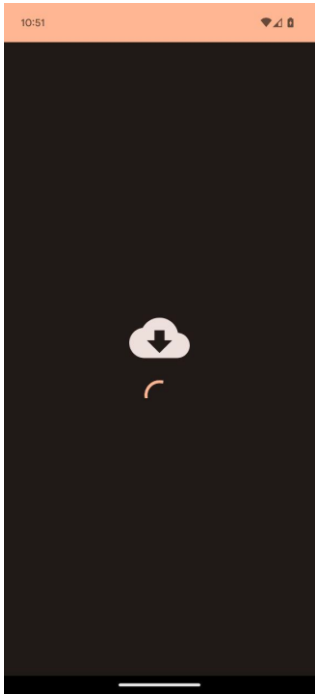

Рисунок 4.3 – Екран завантаження моделі

Також реалізоване діалогове вікно, яке з’являється після того, як користувач виконав дію, спрямовану на припинення процесу завантаження(див. рисунок 4.4). У цьому вікні міститься текст запитання: «Are you sure you want to quit download?», яке уточнює намір користувача. Таке рішення спрямоване на запобігання випадковому завершенню важливих дій. Під текстом розташовано кнопку «Confirm», яка дозволяє завершити завантаження за підтвердженням. Усі елементи цього екрану виконані в темній кольоровій гамі з акцентом на пастельний помаранчевий колір, що виділяє активні елементи інтерфейсу.

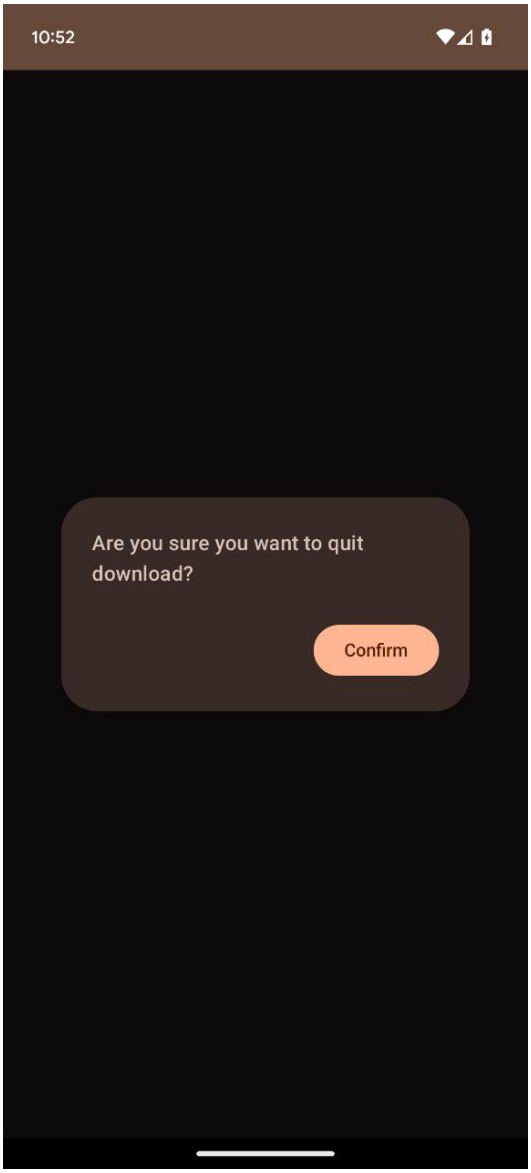

Рисунок 4.4 – Екран з діалогом про вихід

Даний інтерфейс демонструє сучасний підхід до розробки дизайну з акцентом на простоту, мінімалізм та інтуїтивність. Використання універсальних символів і логічне розташування елементів забезпечують зрозумілість процесу навіть для недосвідчених користувачів. Крім того, наявність діалогового вікна підтвердження дій підвищує безпеку і зменшує ймовірність помилкових рішень, що є важливим аспектом у розробці інтуїтивного інтерфейсу.

На рисунку 4.5 представлено базовий етап роботи із застосунком доповненої реальності (AR), де користувач має відсканувати горизонтальну площину для подальшого розміщення об'єктів.

У центрі екрану зображено схематичне зображення смартфона над площиною та текстову інструкцію: "Scan horizontal plane" (Скануйте горизонтальну площину).

Центральне зображення телефону над горизонтальною площиною символізує, що камера пристрою повинна бути спрямована вниз.

Відсутність зайвих деталей на екрані фокусує увагу користувача саме на основному завданні — скануванні.

Мінімалістичний дизайн інструкції допомагає новачкам швидко зрозуміти, що робити.

Екран допомагає зорієнтувати користувача в процесі використання AR. Зображення площини створює уявлення, що застосунок "читає" інформацію з реального світу для побудови доповненої реальності.

Цей етап є критично важливим, адже саме сканування площини забезпечує коректне позиціонування об'єктів.

Схематичний вигляд телефону забезпечує інтуїтивне розуміння: немає складних елементів або тексту, який би ускладнював процес.

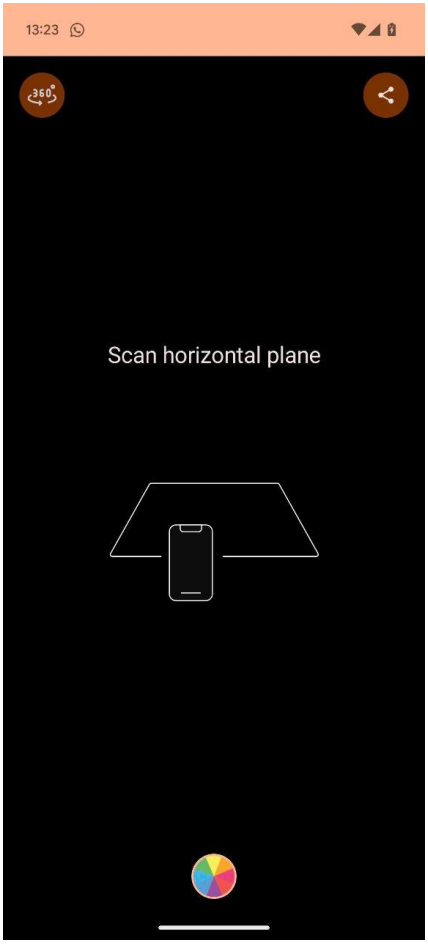

Рисунок 4.5 – Екран з підказкою про сканування поверхні

Екран на рисунку 4.6 з’являється у разі недостатнього освітлення, що є важливою умовою для коректної роботи камер AR. На екрані зображено текст: "Too dark. Try moving to a well-lit area" (Занадто темно. Перейдіть до добре освітленої зони) і "Also, make sure the Block Camera is set to off in system settings" (Переконайтеся, що блокування камери вимкнено в налаштуваннях системи). У центрі екрану схематично показано телефон і площину.

Схема телефону над площиною повторює інструкцію сканування, але текст уточнює проблему — недостатнє освітлення. Чітке повідомлення дозволяє миттєво зрозуміти, що заважає продовжити роботу.

Користувач отримує просте пояснення, чому AR не працює, і одразу розуміє, як виправити ситуацію. Це зменшує ризик розчарування, адже користувач не залишається без пояснень. Лаконічний текст і схема створюють ефективний спосіб вирішення потенційних проблем.

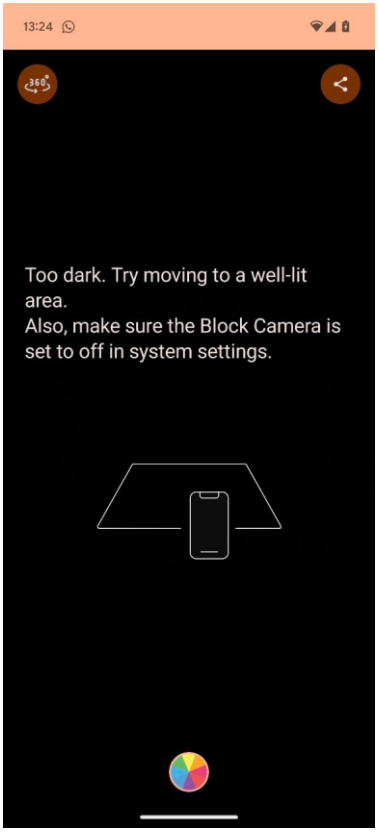

Рисунок 4.6 – Екран з підказкою про освітлення

Наступний екран (рис. 4.7) демонструє, як користувач може взаємодіяти із 3D-моделлю шляхом масштабування. У центрі екрану показано модель (наприклад, стіл), а також анімацію рук, що виконують жест "pinch-to-zoom".

Текстова інструкція пояснює: "Zoom in or out on the model by using a pinch gesture" (Збільшіть або зменшіть модель, використовуючи жест двома пальцями).

Анімація рук із жестом зведення або розведення пальців чітко показує, як користувач може масштабувати об’єкт. 3D-модель об’єкта відображається на площині, щоб користувач бачив прямий зв’язок між діями та результатом.

Цей екран навчає користувача основного способу взаємодії з об’єктами в AR. Жест "pinch-to-zoom" є стандартом у мобільних пристроях, тому його демонстрація створює знайоме відчуття для користувача. Це корисно для перегляду деталей моделі або розуміння її пропорцій у реальному світі.

Анімація забезпечує наочне пояснення жесту, що знижує потребу в текстових інструкціях. Користувач миттєво розуміє, як управляти масштабом.

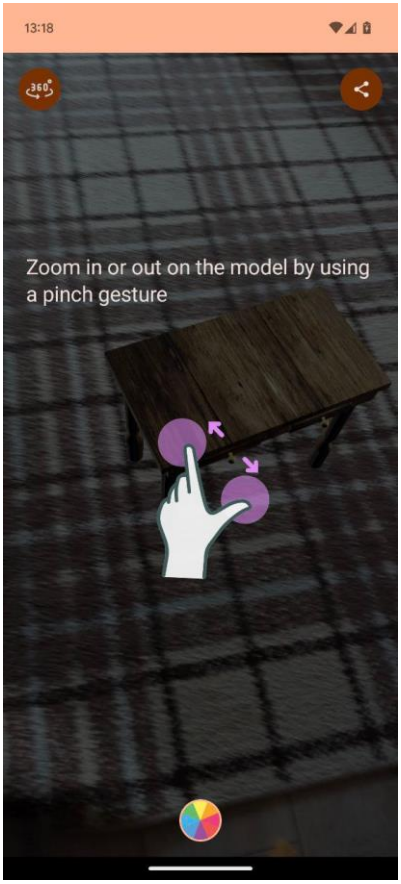

Рисунок 4.7 – Екран з підказкою про приближення моделі в додатку

Наступний екран показує (рис. 4.8), як користувач може розміщувати модель об’єкта на реальній поверхні. У центрі зображено модель меблів і анімацію руки, що переміщує об’єкт. Текстова інструкція: "Move model around the detected surface" (Переміщуйте модель по виявленій поверхні). Нижче з'являється повідомлення: "Model placed successfully" (Модель успішно розміщена).

Анімація руки показує жест, який потрібно виконати для переміщення об’єкта. Об’єкт розташований на поверхні, що дозволяє користувачеві бачити його в реальному масштабі.

Навчання користувача тому, як розміщувати об’єкти, є важливим для використання AR. Текст і анімація працюють разом, щоб забезпечити інтуїтивне розуміння взаємодії. Повідомлення про успішне розміщення моделі підтверджує виконання завдання.

Рисунок 4.8 – Екран з підказкою про можливість рухати об’єкт

Екран на рисунку 4.9 навчає користувача обертати 3D-модель за допомогою жесту "rotate". У центрі екрану зображено анімацію рук, які виконують круговий рух. Текстова інструкція: "Rotate the model by using two fingers" (Обертайте модель, використовуючи два пальці). У нижній частині екрану відображається повідомлення: "Auto rotation stopped" (Автоматичне обертання зупинено).

Анімація демонструє природний спосіб обертання об’єкта за допомогою пальців.

Текст забезпечує додаткове пояснення для тих, хто може не зрозуміти жест.

Екран забезпечує користувача можливістю налаштувати модель відповідно до своїх потреб. Це корисно для перегляду об’єкта з усіх боків, що важливо для оцінки дизайну. Використання анімації жестів полегшує взаємодію, навіть для тих, хто вперше використовує AR.

Рисунок 4.9– Екран з підказкою про можливість повороту об’єкту

На рисунку 4.10 показано скелетон екрану [27] — інтерфейс із заповнювачами, які показують місце майбутнього контенту. Основна частина екрану складається з шести прямокутників, які символізують місця для карток із контентом (зображення чи інформація про об’єкти).

Кожен прямокутник служить місцем для майбутніх елементів контенту, таких як зображення або текст. Їх рівномірне розташування в сітці дає користувачеві зрозуміти, скільки об’єктів буде завантажено. Скелетон екрану відображається під час завантаження даних з сервера. Це запобігає ситуації, коли екран здається порожнім або "зламаним".

Візуальні заповнювачі показують користувачеві, що контент завантажується, створюючи враження швидкості роботи програми, навіть якщо завантаження триває.

Рисунок 4.10 – Екран з шиммером на завантаження товарів

На представленому зображенні (рис. 4.11) зображено екран взаємодії з тривимірною моделлю меблів у середовищі доповненої реальності . Основні функціональні можливості цього екрану — зміна кольору моделі та обертання моделі для перегляду з різних кутів.

У нижній частині екрана розташовані кольорові кола, які представляють різні доступні кольори для моделі:

- Кольори подані у вигляді іконок (наприклад, червоний, сірий, білий, блакитний, рожевий).
- Піктограма з хрестиком дозволяє скинути зміни або повернути модель до базового кольору.

Користувач може торкнутися будь-якого з цих кольорових індикаторів, і модель одразу змінює колір відповідно до вибору.

Ця функція дозволяє користувачеві оцінити, як виглядатимуть меблі в різних кольорових рішеннях у реальному інтер’єрі.

Функція корисна для дизайнерів інтер’єру та користувачів, які підбирають меблі під кольорову гаму приміщення.

Це дозволяє користувачеві експериментувати з дизайном без необхідності фізично змінювати меблі.

У верхньому лівому куті знаходиться кнопка з іконкою "360°", яка дозволяє ввімкнути або змінити режим обертання моделі:

- Натискання на цю кнопку активує функцію обертання.
- Користувач може взаємодіяти з моделлю за допомогою жестів (наприклад, двома пальцями, як показано в попередніх екранах), щоб повертати її та переглядати з різних кутів.

У режимі обертання модель можна рухати в будь-якому напрямку (горизонтально чи вертикально), що дозволяє детально розглянути всі сторони, полиці, шухляди й інші елементи меблів.

Це особливо важливо для користувачів, які хочуть перевірити дизайн меблів або оцінити, як вони виглядатимуть у різних положеннях у кімнаті.

Функція дозволяє уникнути непередбачуваних проблем із дизайном або розміщенням.

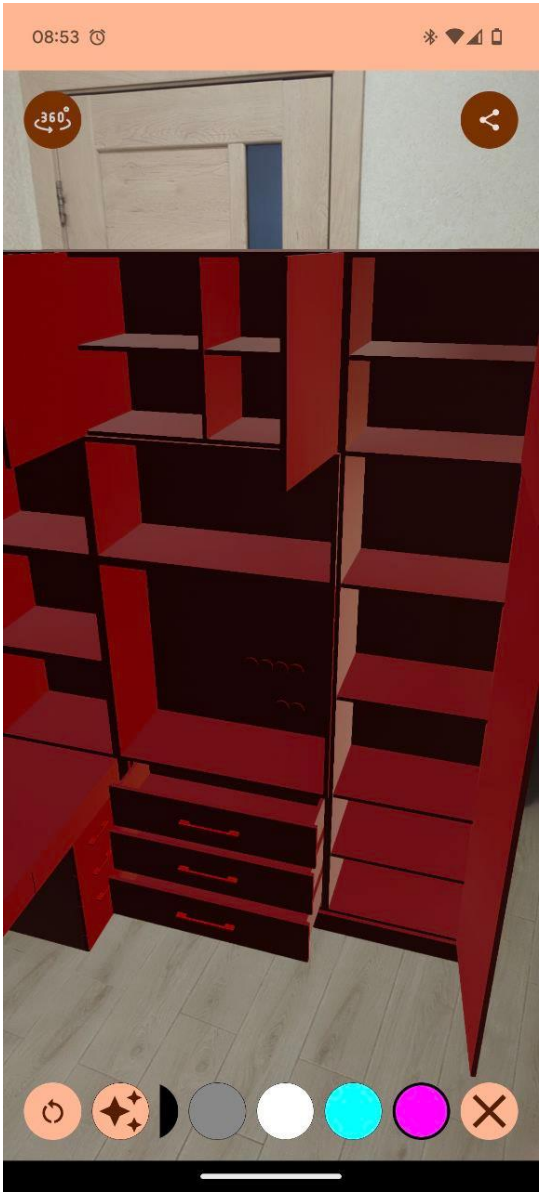

Рисунок 4.11 – Екран з можливістю модифікації об’єкта

На рисунку 4.12 показано екран поширення контенту застосунку, який дозволяє користувачеві поділитися посиланням або зображенням через різні доступні платформи.

Відображаються іконки додатків і контактів, через які користувач може надіслати зображення чи посилання. Імена чи номери телефонів користувачів. Також представлені додактки, Telegram, WhatsApp, Gmail, що забезпечуюсь стандартний обмін повідомленнями тощо.

Додатки розташовані компактно, що дозволяє швидко знайти потрібну платформу.

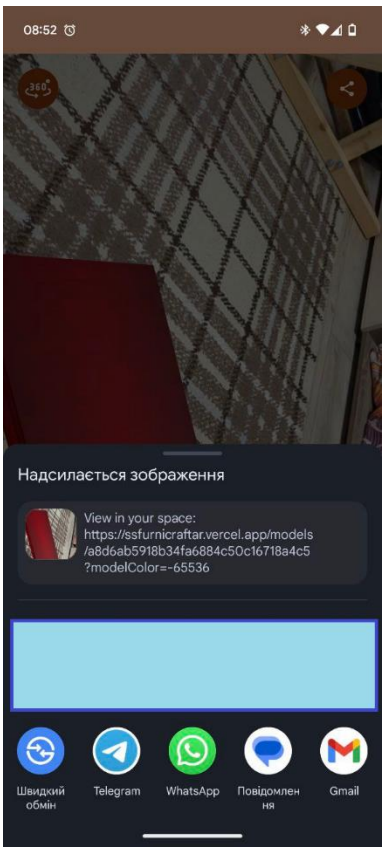

Рисунок 4.12– Екран поширення додатку

4.3 Розробка інструкції користувача

Інструкція для користувача передбачає ознайомлення з мінімальними технічними характеристиками для встановлення застосунку на смартфон. Детальний опис характеристик наведений у таблиці 4.1.

Таблиця 4.1 – Мінімальні технічні характеристики

Характеристика	Опис
Операційна система	Android 9.0 або вище
Процесор	Мінімум 1,22 ГГц, чотирьохядерний або вище
Оперативна пам'ять	Мінімум 1 ГБ RAM
Вільне місце	Мінімум 40 МБ
Інтерфейси	Wi-Fi, LTE

Для запуску мобільного додатку «AR Room» необхідно запустити файл AR Room, зображення іконки в головному меню має вигляд, як наведено на рис. 4.13.

Рисунок 4.13 – Зображення іконки файлу програми в головному меню

4.4 Висновки

У четвертому розділі було проведено тестування програмних модулів мобільної Android-системи для роботи з доповненою реальністю. Були перевірені екрани фільтрації, підказок користувача, завантаження моделей, діалогів, перегляду товарів, також детально протестовано екран роботи з 3D об’єктами.

Розроблено інструкцію користувача з мінімальними характеристиками телефону для встановлення застосунку, а також інструкцію користувача, яка допоможе розпочати використання додатку та швидше ознайомитися зі способами використання функціоналу даного додатку.

Було описано мінімальні технічні характеристики для мобільного пристрою, які забезпечують стабільну та безперебійну роботу додатку.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи .

Магістерська кваліфікаційна робота за темою «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проєкту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5. 1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-

технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [28].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	3	4
2. Ринкові переваги (наявність аналогів)	3	3	4
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3

Продовження таблиці 5.2

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
9. Практична здійсненність (наявність фінансів)	3	4	4
10. Практична здійсненність (необхідність нових матеріалів)	3	4	3
11. Практична здійсненність (термін реалізації)	3	3	3
12. Практична здійсненність (розробка документів)	3	3	3
Сума балів	38	40	41
Середньоарифметична сума балів $СБ_c$	39,7		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3.

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» становить 39,7 бала, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5. 2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними

статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою 5.1:

$$З_о = \sum_{i=1}^{k} \frac{М_{ni} \cdot t_i}{Т_р}, \qquad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

$М_{ni}$ – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

$Т_р$ – середнє число робочих днів в місяці, $Т_р$=24 дні.

$$З_о = 30000 \cdot 96 / 24 = 120000 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проєкту	30000	1250,00	96	120000,00
Інженер-розробник програмного забезпечення	40000	1666,67	90	150000,00
Консультант	18000	750,00	20	15000,00
Всього				285000,00

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними

найменуваннями робіт НДР розраховуємо за формулою 5.2:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \,, \tag{5.2}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою 5.3:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \,, \tag{5.3}$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [29];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 24 дні;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 8000{,}00 \cdot 1{,}10 \cdot 1{,}65 / (24 \cdot 8) = 75{,}63$ грн.

$З_{р1} = 75{,}63 \cdot 6{,}00 = 453{,}75$ грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка електронно-обчислювального обладнання	6	2	1,1	75,63	453,75

Продовження таблиці 5.5

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Підготовка робочого місця дослідника	2,4	2	1,1	75,63	181,50
Заповнення бази даних	12	2	1,7	116,88	1402,50
Перевірка даних на дублікати	3	2	1,1	75,63	226,88
Перевірка додатку на неможливість декодування даних	8	4	1,5	103,13	825,00
Інсталяція програмного забезпечення	4	5	1,7	116,88	467,50
Всього					3557,13

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 5.4:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%.

$З_{дод}$ = (285000,00+ 3557,13) · 10 / 100% =28855,71 грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 5.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (285000,00+ 3557,13+ 28855,71) · 22 / 100% = 69830,82425грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 5.6:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (5.6)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 2,00 · 200,00 · 1,1 - 0,000 · 0,00 = 440,0 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір, уп	200	2	0	0	440
Папір для записів, уп	110	1	0	0	121
Органайзер офісний,шт	210	2	0	0	462
Канцелярське приладдя (набір офісного працівника),шт	175	2	0	0	385
Картридж для принтера	1100	1	0	0	1210
Диск оптичний NewLine CD-RW	15	3	0	0	49,5

Продовження таблиці 5.6

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Flesh-пам'ять Kingston 16 GB	130	1	0	0	143
Тека для паперів	82	2	0	0	180,4
Всього					3490,63

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проєктування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою 5.7:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \quad , \qquad (5.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 300000 \cdot 1 \cdot 1,1 = 330000$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Графічний процесор (GPU) для обчислень	1	300000	330000
Потужний сервер для обробки даних	1	90000	99000
Зарядна станція	1	25000	27500
Всього			456500

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проєктування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою 5.8:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 15000,00 \cdot 3 \cdot 1,1 = 50400$ грн.

Отримані результати зведемо до таблиці:

Таблиця 5.8– Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Операційна система Microsoft Windows 10 Pro for Workstations	3	15000	50400
Всього			50400

При розробці також використовувалось і безкоштовне програмне забезпечення Android Studio.

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою 5.9:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл}$ = (330000,00 · 2) / (3 · 12) = 41250,00 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Графічний процесор (GPU) для обчислень	330000	2	3	41250,00
Потужний сервер для обробки даних	99000	2	3	

Продовження таблиці 5.9

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Зарядна станція	27500	5	3	1375,00
ПК 1	42000	4	3	2625,00
ПК2	33000	2	3	4125,00
Операційна система Microsoft Windows 10 Pro for Workstations	50400	2	3	6300,00
Всього				55675,00

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою 5.10:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \qquad (5.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 11,0 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,9 \cdot 768,0 \cdot 11,0 \cdot 0,95 / 0,97 = 5077,11$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.10 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер 1	0,9	768	5077,11
Робоче місце дослідника	0,15	768	846,19
Пристрої виводу інформації	0,03	0	0,00
Персональний комп'ютер 2	0,9	300	1983,25
Потужний сервер для обробки даних	1,5	500	5509,02
Зарядна станція	0,8	200	1175,26
Всього			14590,82

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 5.11:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \qquad (5.11)$$

де $H_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св}$ = (285000,00+ 3557,13) · 20 / 100% = 57711,43 грн.

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою 5.12:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$В_{сп}$ = (285000,00+ 3557,13) · 30 / 100% = 86567,14 грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 5.13:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де *$Н_{ів}$* – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (285000,00+ 3557,13) · 50 / 100% = 144278,56 грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 5.14:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.14)$$

де *$Н_{нзв}$* – норма нарахування за статтею «Накладні (загальновиробничі)

витрати», приймемо $Н_{нзв}$ = 120%.

$$В_{нзв} = (285000,00+ 3557,13) \cdot 120 / 100\% = 346\ 268,55 \text{ грн.}$$

Витрати на проведення науково-дослідної роботи на тему «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» розраховуємо як суму всіх попередніх статей витрат за формулою 5.15:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.15)$$

$$В_{заг=}\ 1602725,79 \text{ грн.}$$

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою 5.16:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

$$ЗВ = 1602725,79 / 0,9 = 1780806,43 \text{ грн.}$$

5. 3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

Показник	1-й рік	2-й рік	3-й рік
Збільшення кількості споживачів, користувачів	10000	9000	7000

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 10000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 1000,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 50,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою 5.17:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ=38%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (50{,}00 \cdot 10000{,}00 + 1050{,}00 \cdot 10000) \cdot 0{,}83 \cdot 0{,}38 \cdot (1\text{-}0{,}18/100\%) =$ 2855190,8 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (50{,}00 \cdot 10000{,}00 + 1050{,}00 \cdot (10000 + 9000)) \cdot 0{,}83 \cdot 0{,}38 \cdot (1\text{-}0{,}18/100\%) =$

5308059,26 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (50{,}00 \cdot 10000{,}00 + 1050{,}00 \cdot (10000+9000+8000)) \cdot 0{,}83 \cdot 0{,}38 \cdot (1\text{-}0{,}18/100\%) =$

7215845,84 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою 5.18:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,25;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП =$$

$2855190{,}8/(1+0{,}25)^1 + 5308059{,}26/(1+0{,}25)^2 + 7215845{,}84/(1+0{,}25)^3 = 9375823{,}64$ грн

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки, розраховуємо за формулою 5.19:

$$PV = k_{інв} \cdot ЗВ, \qquad (5.19)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження

науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1780806,43 грн.

$PV = k_{інв} \cdot ЗВ$ = 2 · 1780806,43= 3561612,87 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки, розрахований за формулою 5.20, становитиме:

$$E_{абс} = ПП - PV \tag{5.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 9375823,64 грн;

PV – теперішня вартість початкових інвестицій, 3561612,87 грн.

$E_{абс} = ПП - PV$ = 9375823,64 - 3561612,87= 50453194,41грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розрахована за формулою 5.21:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.21}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 50453194,41грн грн;

PV – теперішня вартість початкових інвестицій, 3561612,87грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+50453194,41/3561612,87)$^{1/3}$= 0,38.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ розрахована за формулою 5.22:

$$\tau_{мін} = d + f, \quad (5.22)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,2.

$\tau_{мін}$ = 0,11+0,2 = 0,31 < 0,38 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки розрахований за формулою 5.23:

$$T_{ок} = \frac{1}{E_в}, \quad (5.23)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,38 = 2,6 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5. 4 Висновок

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках» становить 39,7 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

Також термін окупності становить 2,6 років, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках».

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках. Робота оформлена згідно методичних вказівок [30, 31].

Було проаналізовано стан даного питання на сьогоднішній день. Розглянуто основні аналоги, визначено їх особливості та недоліки і зроблено порівняння з власним програмним продуктом. У результаті аналізу обрано мову програмування Kotlin, інтегроване середовище розробки Android Studio.

Розроблено Android-систему «AR Room» для перегляду та взаємодії з 3D об’єктами в режимі доповненої реальньності, яка включає модуль завантаження, модуль відображення 3D об’єктів та анімацій, модуль персоналізованого навчання користувача, модуль для модифікації 3D об’єктів в ході роботи застосунку, модуль фільтрації об’єктів.

Покращено метод інтерактивного відображення даних і взаємодії з користувачем за допомогою доповненої реальності, який використовує анімований оверлей в додатку, що орієнтований на швидке реагування на дії користувача, та покращить користувацький досвід від додатку;

Подальшого розвитку отримав метод створення віртуальних елементів та 3D-об’єктів, який, автоматизує процеси інтеграції об’єктів у віртуальний простір за допомогою додаткового кешування характеристик об’єкту, що вплине на швидкодію додатку та дозволить користуватись ним навіть на девайсах з низькими характеристиками.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка мобільних додатків від А до Я: повний гайд [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/.
2. Войтко В.В. Застосування AR для персоналізованих покупок і вибору товарів в android-додатках / В. В. Войтко, Д.М. Карабіньовський, А.В.Денисюк // Інформаційні технології і автоматизація – 2024 / Матеріали XVII міжнародної науково-практичної конференції. Одеса, 31 жовтня – 1 листопада 2024 р. – Одеса, Видавництво ОНТУ, 2024 р. – С. 451-453.
3. Android [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Android
4. Augmented Reality [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Доповнена_реальність
5. Pokemon Go [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Pokémon_Go
6. IKEA Place [Електронний ресурс] – Режим доступу до ресурсу: https://www.ikea.com/global/en/newsroom/innovation/ikea-launches-ikea-place-a-new-app-that-allows-people-to-virtually-place-furniture-in-their-home-170912/
7. Google Lens [Електронний ресурс] – Режим доступу до ресурсу: https://lens.google
8. Snapchat [Електронний ресурс] – Режим доступу до ресурсу: https://www.snapchat.com
9. Quiver [Електронний ресурс] – Режим доступу до ресурсу: https://quivervision.com
10. React Native [Електронний ресурс] – Режим доступу до ресурсу: https://reactnative.dev
11. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org
12. ARCore [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/ar?hl=ua

13. Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/compose
14. State and Jetpack Compose [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/jetpack/compose/state.
15. Jetpack Compose Preview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/develop/ui/compose/tooling/previews
16. MVVM [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/topic/libraries/architecture/viewmodel
17. GLTF [Електронний ресурс] – Режим доступу до ресурсу: https://docs.blender.org/manual/uk/2.82/addons/import_export/scene_gltf2.html
18. Creating libraries and .aar explanation [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/projects/android-library
19. Принцип єдиної відповідальності [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Принцип_єдиної_відповідальності
20. UML [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language
21. Діаграма взаємодії [Електронний ресурс] – https://uk.wikipedia.org/wiki/Діаграма_взаємодії
22. Lottie animations and how to use them [Електронний ресурс] – Режим доступу до ресурсу: https://cases.media/en/learning/lecture/sho-take-lottie-animaciya-ta-yak-vona-vikoristovuyetsya?srsltid=AfmBOorV4WnjrcscFbOV6nLigAS4RDi15WK94vZXuhMc2DUuQt-Dr5GZ
23. Black box testing [Електронний ресурс] – Режим доступу до ресурсу: https://mate.academy/blog/qa/black-box-testing/
24. White box testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-white-box-testing/#:~:text=White%20box%20testing%20is%20a,gray%20and%20black%20box%20testing.

25. Grey box testing [Електронний ресурс] – https://www.imperva.com/learn/application-security/gray-box-testing/#:~:text=Gray%20box%20testing%20(a.k.a%20grey,of%20the%20component%20being%20tested.
26. Splash screen [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Splash_screen#:~:text=A%20splash%20screen%20is%20a,introduction%20page%20on%20a%20website.
27. Skeleton screen [Електронний ресурс] – Режим доступу до ресурсу: https://mailchimp.com/resources/skeleton-screen/#:~:text=Skeleton%20screens%20are%20a%20user,continuity%20in%20the%20user%20interface.
28. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.
29. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.
30. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А.О. Семенов, Л.П. Громова, Т.В. Макарова, О.В. Сердюк. Вінниця : ВНТУ, 2021. – 60 с.
31. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 50 с.

ДОДАТКИ

112

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«19» вересня 2024 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доц. Г.О. Черноволик
«19» вересня 2024 року.

Виконав:
студент гр. 1ПІ-23м Д.М. Карабіньовський
«19» вересня 2024 року.

м. Вінниця – 2024 рік

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках». Галузь застосування – мобільні системи.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №310 від 17 вересня 2024 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення інтерактивності мобільних Android-додатків шляхом розробки і впровадження методів і засобів використання доповненої реальності, що дозволить розширити функціональні можливості системи.

Призначення роботи – Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Розробка мобільних додатків від А до Я: повний гайд [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/.
2. Kotlin coroutines on Android [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/kotlin/coroutines.
3. Романюк О.Н. Організація баз даних і знань. / О.Н. Романюк, Т.О. Савчук //Навчальний посібник. – Вінниця: УНІВЕРСУМ –Вінниця. –2003. – 123 с.

5. Технічні вимоги

Вхідні дані до роботи – Android-додаток, доповнена реальність, мобільні додатки, 3D-моделювання, тестування продуктивності; вихідні дані – графічний

інтерфейс з можливістю взаємодії з 3D об’єктами в режимі віртуальної реальності.

6. Конструктивні вимоги.

Інтерфейс програми повинен відповідати естетичним та ергономічним вимогам та мати зручну навігацію і засоби керування.

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку систем доповненої реальності в андроїд застосунках та постановка задач дослідження	20.09.2024 30.09.2024
2	Розробка методів та моделей Android-системи	01.10.2024 17.10.2024
3	Розробка програми	18.10.2024 04.11.2024
4	Тестування програми	05.11.2024 21.11.2024
5	Економічна частина	22.11.2024 30.11.2024

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б
Протокол перевірки на плагіат
ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках**

Тип роботи: кваліфікаційна робота
Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ-23м
Науковий керівник: к.т.н., доц. каф. ПЗ Черноволик Г. О.

Turnitin	
Оригінальність	96%
Схожість	4%

Аналіз звіту подібності

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

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

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

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках».

Автор _____ Карабіньовський Даниїл Максимович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт (за потреби) посада) _____ (підпис) _____ (прізвище, ініціали,

Додаток В – Лістинг програми

```
@HiltAndroidApp
class SSFurniCraftARApplication : Application() {

    override fun onCreate() {
        super.onCreate()

        initTimber()
    }

    private fun initTimber() {
        if (BuildConfig.DEBUG) {
            // Show logs only when on debug
            Timber.plant(Timber.DebugTree())
        }
    }
}


@Composable
internal fun ARCoachingOverlay(
    modifier: Modifier = Modifier,
    message: String = ""
) {
    val composition by
rememberLottieComposition(LottieCompositionSpec.RawRes(R.raw.ar_coaching_overlay))
    val progress by animateLottieCompositionAsState(
        composition = composition,
        iterations = LottieConstants.IterateForever
    )

    Column(
        modifier = modifier
            .padding(LocalDimens.SpacingLarge)
            .fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        Text(
            text = message,
            style = MaterialTheme.typography.titleLarge
        )

        LottieAnimation(
            composition = composition,
            progress = { progress }
        )
    }
}
private lateinit var dynamicBitmap: Bitmap
private const val DYNAMIC_COLOR_UPDATE_THRESHOLD = 400 // ms
```

```
@Composable
internal fun ARView(
    modifier: Modifier = Modifier,
    arViewUiState: ARViewUiState,
    rotationEnabled: Boolean,
    onStopRotation: () -> Unit,
    enableCapture: Boolean = false,
    onCapture: ((Bitmap) -> Unit)? = null,
    onModelPlace: () -> Unit
) {
    // YUV image to RGB image converter
    val rgbConverter = rememberYuvToRgbConverter()

    // The destroy calls are automatically made when their disposable effect leaves
    // the composition or its key changes.
    val engine = rememberEngine()
    val modelLoader = rememberModelLoader(engine)
    val childNodes = rememberNodes()
    val view = rememberView(engine)

    val cameraNode = remember { ARSceneView.createARCameraNode(engine) }


    var model by remember { mutableStateOf<ModelNode?>(null) }


    var planeRenderer by remember { mutableStateOf(true) }


    var trackingFailureReason by remember {
        mutableStateOf<TrackingFailureReason?>(null)
    }
    var frame by remember { mutableStateOf<Frame?>(null) }


    var arSceneView by remember { mutableStateOf<ARSceneView?>(null) }

    var animateModel by remember { mutableStateOf(false) }
    var originalMaterials by remember { mutableStateOf<List<List<MaterialInstance>>?>(null) }
    var colorMaterials by remember { mutableStateOf<List<List<MaterialInstance>>?>(null) }
    var showCoachingOverlay by remember { mutableStateOf(true) }
    var showGestureOverlay by remember { mutableStateOf(false) }
    val isDynamicColor by remember(arViewUiState.modelColor) {
        derivedStateOf {
            arViewUiState.modelColor is ColorState.Dynamic
        }
    }
    var currentColor by remember { mutableStateOf(Color.White) }
    val animatedColor by animateColorAsState(
        targetValue = currentColor,
        label = "ModelAnimatedColor",
```

```
        animationSpec = tween(DYNAMIC_COLOR_UPDATE_THRESHOLD, easing =
LinearEasing)
    )
    val lastColorUpdateTime by remember(currentColor) {
        mutableLongStateOf(System.currentTimeMillis())
    }

    LaunchedEffect(key1 = animatedColor) {
        colorMaterials?.setColor(animatedColor.toSVColor())
    }

    LaunchedEffect(key1 = enableCapture) {
        if (enableCapture) {
            arSceneView?.run { onCapture?.invoke(captureImage()) }
        }
    }

    LaunchedEffect(key1 = arViewUiState.modelPath) {
        val path = arViewUiState.modelPath

        model = modelLoader.loadModelInstance(path)?.let { instance ->
            ModelNode(
                modelInstance = instance
            ).apply {
                // Change models initial scale. Currently using width
                scale = Scale(Constants.MODEL_INITIAL_SIZE / size.x)
                // Reset model's initial rotation
                rotation = Rotation()
                // Enable custom gestures on model
                enableGestures()
                // Keep reference to original material to reset color
                originalMaterials = materialInstances
                // Clone materials to apply new colors
                colorMaterials = materialInstances.clone()
            }
        }
    }

    LaunchedEffect(key1 = animateModel) {
        model?.apply {
            if (animateModel) startBouncingEffect() else endBouncingEffect()
        }
    }

    LaunchedEffect(key1 = rotationEnabled) {
        if (rotationEnabled) {
            model?.startModelRotation()
        }
    }

    LaunchedEffect(key1 = arViewUiState.modelColor, key2 = model) {
        when (arViewUiState.modelColor) {
```

```
            is ColorState.Color -> {
                colorMaterials?.let { model?.materialInstances = it }
                currentColor = arViewUiState.modelColor.value
            }

            ColorState.Dynamic -> {
                colorMaterials?.let { model?.materialInstances = it }
            }

            ColorState.None -> {
                originalMaterials?.let { model?.materialInstances = it }
                currentColor = Color.White
            }
        }
    }

    Box(modifier = modifier.fillMaxSize()) {
        ARScene(
            modifier = modifier.fillMaxSize(),
            cameraNode = cameraNode,
            childNodes = childNodes,
            engine = engine,
            view = view,
            modelLoader = modelLoader,
            onGestureListener = rememberOnGestureListener(
                onSingleTapConfirmed = { _, _ ->
                    if (animateModel) {
                        onModelPlace()
                        animateModel = false
                        showGestureOverlay = true
                    }
                    onStopRotation()
                },
                onMove = { _, event, node ->
                    if (node == null) return@rememberOnGestureListener
                    // Move gesture to move model node. Instead of changing model position
                    // change anchor node position (parent of model node)
                    frame?.hitTest(event)?.firstOrNull()?.hitPose?.position?.let { position ->
                        val anchor = (node.parent as? AnchorNode) ?: node
                        anchor.worldPosition = position
                    }
                }
            ),
            sessionCameraConfig = { session -> session.disableDepthConfig() },
            sessionConfiguration = { _, config -> config.configure() },
            planeRenderer = planeRenderer,
            onTrackingFailureChanged = {
                showCoachingOverlay = it != null
                trackingFailureReason = it
            },
            onSessionUpdated = { session, updatedFrame ->
                frame = updatedFrame
```

```
            if (session.config.planeFindingMode != arViewUiState.findingMode) {
                session.configure {
                    it.planeFindingMode = arViewUiState.findingMode
                    Timber.d("Updating finding mode to: ${it.planeFindingMode}")
                }
            }

            // If no model is placed then create anchor node and add
            // model node to that anchor
            if (childNodes.isEmpty()) {
                updatedFrame.createCenterAnchorNode(engine)?.let { anchorNode ->
                    model?.let {
                        animateModel = true
                        anchorNode.addChildNode(it)
                        childNodes.add(anchorNode)
                        planeRenderer = false
                        showCoachingOverlay = false
                    }
                }
            } else if (animateModel) {
                (childNodes.firstOrNull() as? AnchorNode)?.apply {
                    val (x, y) = view.viewport.run {
                        with(LocalDimens.ARView) {
                            width / MODEL_PLACEMENT_WIDTH_PROPORTION to height /
MODEL_PLACEMENT_HEIGHT_PROPORTION
                        }
                    }

                    val newPosition = updatedFrame.getPose(x, y)?.position ?: return@ARScene
                    worldPosition = Position(newPosition.x, newPosition.y, newPosition.z)
                }
            }

            if (isDynamicColor) {
                if (System.currentTimeMillis() - lastColorUpdateTime <
DYNAMIC_COLOR_UPDATE_THRESHOLD) return@ARScene
                currentColor = updatedFrame.getDynamicColor(rgbConverter) ?: return@ARScene
            }
        },
        onViewCreated = { arSceneView = this },
        onViewUpdated = { arSceneView = this }
    )

    AnimatedVisibility(
        modifier = Modifier
            .align(Alignment.Center)
            .background(Color.Black.copy(alpha = LocalDimens.ARView.OverlayAlpha)),
        visible = showCoachingOverlay,
        enter = fadeIn(),
        exit = fadeOut()
    ) {
```

```
            ARCoachingOverlay(
                message = trackingFailureReason?.getDescription(LocalContext.current)
                    ?: arViewUiState.findingMode.getDescription(LocalContext.current)
            )
        }

        AnimatedVisibility(
            modifier = Modifier
                .align(Alignment.Center)
                .background(Color.Black.copy(alpha = LocalDimens.ARView.OverlayAlpha)),
            visible = showGestureOverlay,
            enter = fadeIn(),
            exit = fadeOut()
        ) {
            GestureOverlay {
                showGestureOverlay = false
            }
        }
    }
}

/**
 * Disable Depth API
 * Depth API may give better results. But, some devices (tested on Samsung S20+)
 * creates too much lag. Reason is unknown. Looks like the image acquired
 * with `frame.acquireCameraImage()` is not realised. so we won't be able to acquire
 * new frames when the rate limit is exceeded.
 */
private fun Session.disableDepthConfig(): CameraConfig {
    val filter = CameraConfigFilter(this)
    filter.setDepthSensorUsage(EnumSet.of(CameraConfig.DepthSensorUsage.DO_NOT_USE))
    val cameraConfigList = getSupportedCameraConfigs(filter)
    return cameraConfigList.first()
}

/**
 * Configurations for ARSession
 */
private fun Config.configure() {
    depthMode = Config.DepthMode.DISABLED
    updateMode = Config.UpdateMode.LATEST_CAMERA_IMAGE
    instantPlacementMode = Config.InstantPlacementMode.LOCAL_Y_UP
    planeFindingMode = PlaneFindingMode.HORIZONTAL
    lightEstimationMode = Config.LightEstimationMode.ENVIRONMENTAL_HDR
}

/**
 * Create anchor node at the center of the screen
 */
private fun Frame.createCenterAnchorNode(engine: Engine): AnchorNode? =
    getUpdatedPlanes().firstOrNull()
        ?.let { it.createAnchorOrNull(it.centerPose) }
```

```
        ?.let { anchor ->
            AnchorNode(engine, anchor).apply {
                isEditable = false
                isPositionEditable = false
                updateAnchorPose = false
            }
        }

/**
 * Get real world position from the given [x] & [y] coordinates.
 */
private fun Frame.getPose(x: Float, y: Float): Pose? =
    hitTest(x, y).firstOrNull()?.hitPose

/**
 * Get dynamic color from receiver [Frame].
 *
 * @return Returns dynamic [Color] or null if failed to get.
 */
private fun Frame.getDynamicColor(converter: YuvToRgbConverter): Color? =
    try {
        acquireCameraImage().use { image ->
            if (!::dynamicBitmap.isInitialized) {
                dynamicBitmap =
                    Bitmap.createBitmap(image.width, image.height, Bitmap.Config.ARGB_8888)
            }
            converter.yuvToRgb(image, dynamicBitmap)
        }

        val color = dynamicBitmap.getVibrantColor()
        // If failed to get the vibrant color, don't return default color
        if (color == 0) null else Color(color)
    } catch (notAvailableException: NotYetAvailableException) {
        Timber.d("Frame is not yet available")
        null
    }
@Composable
internal fun GestureOverlay(
    modifier: Modifier = Modifier,
    iterations: Int = Constants.DEFAULT_GESTURE_OVERLAY_ITERATIONS,
    onComplete: () -> Unit
) {
    // Create lottie composition specs for all gestures
    val specs = listOf(
        R.raw.anim_move_gesture,
        R.raw.anim_rotate_gesture,
        R.raw.anim_zoom_gesture
    ).map(LottieCompositionSpec::RawRes)

    // Display messages for gesture
    val messages = listOf(
        R.string.message_move_gesture,
```

```
        R.string.message_rotate_gesture,
        R.string.message_zoom_gesture
    ).map { stringResource(it) }

    // Number of times the gestures are played
    var playedCount by rememberSaveable { mutableIntStateOf(Constants.PLAY_COUNT_ZERO)
}

    // Current index of gesture
    val currentGestureIndex by remember(playedCount) {
        // If each gesture is already played for [iterations] times, call onComplete
        if (playedCount >= iterations * specs.size) {
            onComplete()
        }
        mutableIntStateOf(playedCount % specs.size)
    }

    Column(
        modifier = modifier
            .padding(LocalDimens.SpacingLarge)
            .fillMaxSize(),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        key(currentGestureIndex) {
            Text(
                text = messages[currentGestureIndex],
                style = MaterialTheme.typography.titleLarge
            )

            LottieView(compositionSpec = specs[currentGestureIndex]) {
                playedCount++
            }
        }
    }
}

@Composable
private fun LottieView(
    modifier: Modifier = Modifier,
    compositionSpec: LottieCompositionSpec,
    onEnd: () -> Unit
) {
    val composition by rememberLottieComposition(compositionSpec)
    val progress by animateLottieCompositionAsState(composition)
    LottieAnimation(
        modifier = modifier.aspectRatio(LocalDimens.ARView.GestureAnimRatio),
        composition = composition,
        contentScale = ContentScale.FillWidth,
        progress = {
            if (progress == Constants.ANIM_COMPLETE_VALUE) {
                onEnd()
```

```
            }
            progress
        }
    )
}
@Composable
internal fun Options(
    modifier: Modifier = Modifier,
    arViewUiState: ARViewUiState,
    rotationEnabled: Boolean,
    onRotationToggle: () -> Unit,
    onShare: () -> Unit,
    onColorChange: (ColorState) -> Unit
) {
    Box(
        modifier = modifier
            .fillMaxSize()
            .padding(LocalDimens.SpacingMedium)
    ) {
        RotationOption(
            rotationEnabled = rotationEnabled,
            onClick = onRotationToggle
        )

        ShareOption(
            modifier = Modifier.align(Alignment.TopEnd),
            onShare = onShare
        )

        ColorOption(
            modifier = Modifier.align(Alignment.BottomCenter),
            selectedColor = arViewUiState.modelColor,
            onSelect = onColorChange
        )
    }
}

@Composable
fun RotationOption(
    modifier: Modifier = Modifier,
    rotationEnabled: Boolean,
    onClick: () -> Unit,
) {
    val contentColor by animateColorAsState(
        targetValue = if (rotationEnabled) MaterialTheme.colorScheme.primary else Color.LightGray,
        label = "RotationIconColor"
    )

    Icon(
        modifier = modifier
            .size(LocalDimens.ARView.OptionsIconSize)
            .clip(CircleShape)
```

```
            .background(MaterialTheme.colorScheme.primaryContainer)
            .clickable { onClick() }
            .padding(LocalDimens.SpacingSmall),
        painter = painterResource(R.drawable.ic_rotate_360),
        contentDescription = stringResource(R.string.cd_rotate_360),
        tint = contentColor
    )
}

@Composable
private fun ShareOption(
    modifier: Modifier = Modifier,
    onShare: () -> Unit
) {
    Icon(
        modifier = modifier
            .size(LocalDimens.ARView.OptionsIconSize)
            .clip(CircleShape)
            .background(MaterialTheme.colorScheme.primaryContainer)
            .clickable { onShare() }
            .padding(LocalDimens.ARView.ShareOptionSpacing),
        imageVector = Icons.Default.Share,
        contentDescription = stringResource(R.string.share)
    )
}

@Composable
private fun ColorOption(
    modifier: Modifier = Modifier,
    selectedColor: ColorState,
    onSelect: (ColorState) -> Unit
) {
    var showPicker by rememberSaveable { mutableStateOf(false) }

    BoxWithConstraints(modifier = modifier) {
        Row(
            modifier = Modifier
                .height(LocalDimens.ARView.OptionsIconSize)
                .clip(CircleShape)
        ) {
            AnimatedVisibility(showPicker) {
                Row(
                    modifier = Modifier
                        .widthIn(
                            max = this@BoxWithConstraints.maxWidth -
LocalDimens.ARView.OptionsIconSize
                        )
                        .padding(end = LocalDimens.SpacingSmall),
                    horizontalArrangement = Arrangement.spacedBy(LocalDimens.SpacingSmall)
                ) {
                    Button(
                        modifier = Modifier.size(LocalDimens.ARView.OptionsIconSize),
```

```
                onClick = {
                    onSelect(ColorState.None)
                },
                contentPadding = PaddingValues(LocalDimens.NoSpacing)
            ) {
                Icon(
                    imageVector = Icons.Default.RestartAlt,
                    contentDescription = stringResource(R.string.reset_color)
                )
            }

            DynamicColorButton(
                isSelected = selectedColor is ColorState.Dynamic
            ) {
                onSelect(ColorState.Dynamic)
            }

            ColorPicker(
                modifier = Modifier,
                initialColor = (selectedColor as? ColorState.Color)?.value,
                onSelect = { onSelect(ColorState.Color(it)) }
            )
        }
    }

    Button(
        modifier = Modifier.size(LocalDimens.ARView.OptionsIconSize),
        onClick = { showPicker = !showPicker },
        contentPadding = PaddingValues(LocalDimens.NoSpacing)
    ) {
        val iconDescription = stringResource(R.string.color_picker)
        AnimatedContent(targetState = showPicker, label =
Constants.ANIM_LABEL_COLOR_PICKER) { showPicker ->
            if (showPicker) {
                Icon(
                    modifier = Modifier.fillMaxSize(),
                    imageVector = Icons.Default.Close,
                    contentDescription = iconDescription,
                )
            } else {
                Image(
                    painter = painterResource(R.drawable.ic_color_picker),
                    contentDescription = iconDescription,
                    contentScale = ContentScale.FillBounds
                )
            }
        }
    }
  }
}
```

```
@Composable
private fun DynamicColorButton(
    modifier: Modifier = Modifier,
    isSelected: Boolean = false,
    onSelect: () -> Unit
) {
    Icon(
        modifier = modifier
            .clip(CircleShape)
            .background(MaterialTheme.colorScheme.primary)
            .size(LocalDimens.ARView.OptionsIconSize)
            .border(
                width = if (isSelected) 2.dp else 0.dp,
                color = Color.Black,
                shape = CircleShape
            )
            .padding(LocalDimens.SpacingXS)
            .clickable { onSelect() },
        imageVector = Icons.Default.AutoAwesome,
        contentDescription = stringResource(R.string.dynamic_color),
        tint = MaterialTheme.colorScheme.onPrimary
    )

}
internal const val PRODUCT_ID_ARG = "productId"
internal const val MODEL_COLOR_ARG = "modelColor"
const val ARVIEW_ROUTE_BASE = "arview_route"
const val ARVIEW_ROUTE =
"$ARVIEW_ROUTE_BASE/{$PRODUCT_ID_ARG}?$MODEL_COLOR_ARG={$MODEL_C
OLOR_ARG}"

fun NavController.navigateToARView(productId: String, modelColor: ColorState =
ColorState.None) {
    val baseRoute = "$ARVIEW_ROUTE_BASE/$productId"
    val colorValue = modelColor.stringValue
    val route = colorValue?.let { "$baseRoute?$MODEL_COLOR_ARG=$it" } ?: baseRoute
    navigate(route)
}

fun NavGraphBuilder.arViewScreen(
    onNavigateBack: () -> Unit,
    onShowSnackbar: suspend (String, String?) -> Boolean
) {
    composable(
        route = ARVIEW_ROUTE,
        arguments = listOf(
            navArgument(PRODUCT_ID_ARG) { type = NavType.StringType },
            navArgument(MODEL_COLOR_ARG) {
                type = NavType.StringType
                nullable = true
                defaultValue = null
            }
```

```
        )
    ) {
        ARViewRoute(onNavigateBack = onNavigateBack, onShowSnackbar = onShowSnackbar)
    }
}

internal data class ARViewArgs(
    val productId: String,
    val modelColor: ColorState
) {
    constructor(savedStateHandle: SavedStateHandle) : this(
        productId = requireNotNull(savedStateHandle[PRODUCT_ID_ARG]),
        modelColor =
ColorState.parseFrom(savedStateHandle.get<String>(MODEL_COLOR_ARG))
    )
}
@Composable
fun ARViewRoute(
    modifier: Modifier = Modifier,
    onNavigateBack: () -> Unit,
    onShowSnackbar: suspend (String, String?) -> Boolean,
    viewModel: ARViewViewModel = hiltViewModel()
) {
    val arViewUiState by viewModel.arViewUiState.collectAsStateWithLifecycle()

    ARViewScreen(
        modifier = modifier,
        arViewUiState = arViewUiState,
        onColorChange = viewModel::changeColor,
        onShare = viewModel::createShareUri,
        onNavigateBack = onNavigateBack,
        onShowSnackbar = onShowSnackbar
    )
}

@Composable
private fun ARViewScreen(
    modifier: Modifier = Modifier,
    arViewUiState: ARViewUiState,
    onColorChange: (ColorState) -> Unit,
    onShare: (Bitmap) -> Unit,
    onNavigateBack: () -> Unit,
    onShowSnackbar: suspend (String, String?) -> Boolean
) {
    var showQuitDialog by remember { mutableStateOf(false) }
    var captureImage by rememberSaveable {
        mutableStateOf(false)
    }
    var capturedImage by remember {
        mutableStateOf<Bitmap?>(null)
    }
```

```
val coroutineScope = rememberCoroutineScope()
val context = LocalContext.current
val shareMessage = stringResource(
    R.string.message_share,
    Urls.getModelUrl(arViewUiState.productId, arViewUiState.modelColor)
)
val modelPlacedMessage = stringResource(R.string.model_placed_successfully)

val dynamicColorMessage = stringResource(R.string.dynamic_color_enabled)

var rotationEnabled by rememberSaveable { mutableStateOf(false) }
var showRotationChangeMessage by rememberSaveable { mutableStateOf(false) }
val rotationStartedMessage = stringResource(R.string.message_auto_rotation_started)
val rotationStoppedMessage = stringResource(R.string.message_auto_rotation_stopped)

LaunchedEffect(key1 = capturedImage) {
    capturedImage?.let { bitmap ->
        captureImage = false
        onShare(bitmap)
    }
}

LaunchedEffect(key1 = arViewUiState.shareUri) {
    arViewUiState.shareUri?.let { uri ->
        context.shareImage(uri, shareMessage)
    }
}

LaunchedEffect(key1 = arViewUiState.modelColor) {
    if (arViewUiState.modelColor is ColorState.Dynamic) {
        onShowSnackbar(dynamicColorMessage, null)
    }
}

LaunchedEffect(key1 = showRotationChangeMessage, key2 = rotationEnabled) {
    if (showRotationChangeMessage) {
        onShowSnackbar(
            if (rotationEnabled) rotationStartedMessage else rotationStoppedMessage,
            null
        )
        showRotationChangeMessage = false
    }
}

BackHandler {
    showQuitDialog = !showQuitDialog
}

Box(modifier = modifier) {
    ARView(
        arViewUiState = arViewUiState,
        rotationEnabled = rotationEnabled,
```

```
            onStopRotation = {
                if (rotationEnabled) {
                    rotationEnabled = false
                    showRotationChangeMessage = true
                }
            },
            enableCapture = captureImage,
            onCapture = { capturedImage = it },
        ) {
            coroutineScope.launch {
                onShowSnackbar(modelPlacedMessage, null)
            }
        }

        Options(
            rotationEnabled = rotationEnabled,
            onRotationToggle = {
                rotationEnabled = !rotationEnabled
                showRotationChangeMessage = true
            },
            onShare = { captureImage = true },
            arViewUiState = arViewUiState,
            onColorChange = onColorChange
        )

        if (showQuitDialog) {
            QuitDialog(
                message = stringResource(R.string.title_arview_quit),
                onDismiss = { showQuitDialog = false }
            ) {
                showQuitDialog = false
                onNavigateBack()
            }
        }
    }
}
data class ARViewUiState(
    val productId: String,
    val modelPath: String,
    val findingMode: Config.PlaneFindingMode = Config.PlaneFindingMode.HORIZONTAL,
    val modelColor: ColorState = ColorState.None,
    val shareUri: Uri? = null
)
@HiltViewModel
class ARViewViewModel @Inject constructor(
    savedStateHandle: SavedStateHandle,
    private val fileHelper: FileHelper,
    private val getProductCategoryUseCase: GetProductCategoryUseCase
) : ViewModel() {

    private val args = ARViewArgs(savedStateHandle)
```

```
    private val _arViewUiState = MutableStateFlow(
        ARViewUiState(
            productId = args.productId,
            modelPath = getModelPath(args.productId).asStringPath(),
            modelColor = args.modelColor
        )
    )
    val arViewUiState = _arViewUiState.asStateFlow()

    init {
        updatePlaneFindingMode(args.productId)
    }

    fun changeColor(color: ColorState) {
        _arViewUiState.update { it.copy(modelColor = color) }
    }

    fun createShareUri(bitmap: Bitmap) {
        val fileName =
"${Calendar.getInstance().timeInMillis}.${Constants.IMAGE_FILE_EXTENSION}"
        val filePath = fileHelper.imageShareDir() / Paths.get(fileName)

        viewModelScope.launch {
            if (bitmap.saveToFile(filePath)) {
                _arViewUiState.update {
                    it.copy(shareUri = fileHelper.getUriForFile(filePath.toFile()))
                }
            }
        }
    }

    /**
     * Get local model path from given [productId].
     */
    private fun getModelPath(productId: String): Path =
        fileHelper.getModelDir().resolve("${productId}.glb")

    /**
     * Update plane finding mode for given [productId]
     */
    private fun updatePlaneFindingMode(productId: String) {
        viewModelScope.launch {
            val findingMode = getProductCategoryUseCase(productId).planeType.arFindingMode
            _arViewUiState.update { it.copy(findingMode = findingMode) }
        }
    }

    private fun Path.asStringPath(): String = toUri().toString()
}
sealed interface ColorState {

    data object None: ColorState
```

```
    data class Color(val value: ComposeColor): ColorState

    data object Dynamic: ColorState

    val stringValue: String?
        get() = when(this) {
            is Color -> value.toArgb().toString()
            Dynamic -> DYNAMIC_COLOR
            None -> null
        }

    companion object {
        const val DYNAMIC_COLOR = "dynamic"

        fun parseFrom(value: String?): ColorState {

            if (value == DYNAMIC_COLOR) return Dynamic

            val colorValue = value?.toIntOrNull()?.let { Color(ComposeColor(it)) }
            return colorValue ?: None
        }
    }
}
private val colors = with(Color) {
    listOf(Red, Green, Blue, Yellow, Black, Gray, White, Cyan, Magenta)
}

@Composable
fun ColorPicker(
    modifier: Modifier = Modifier,
    initialColor: Color? = null,
    onSelect: (Color) -> Unit
) {
    var currentColor by rememberSaveable(initialColor, stateSaver = ColorSaver) {
        mutableStateOf(initialColor)
    }

    Row(
        modifier = modifier.horizontalScroll(rememberScrollState()),
        horizontalArrangement = Arrangement.spacedBy(LocalDimens.SpacingSmall)
    ) {
        colors.forEach { color ->
            val isSelected = currentColor == color
            Box(
                modifier = Modifier
                    .padding(LocalDimens.NoSpacing)
                    .size(LocalDimens.ARView.OptionsIconSize)
                    .clip(CircleShape)
                    .border(
                        width = with(LocalDimens.ColorPicker) {
                            if (isSelected) SelectionBorderSize else NoSelectionBorderSize
```

```
                    },
                    color = Color.Black,
                    shape = CircleShape
                )
                .background(color)
                .clickable {
                    currentColor = color
                    onSelect(color)
                }
        )
    }
  }
}
@Composable
fun QuitDialog(
  modifier: Modifier = Modifier,
  message: String,
  onDismiss: () -> Unit,
  onConfirm: () -> Unit
) {
  AlertDialog(
    modifier = modifier,
    onDismissRequest = onDismiss,
    confirmButton = {
      Button(onClick = onConfirm) {
        Text(text = stringResource(id = R.string.confirm))
      }
    },
    text = {
      Text(
        text = message,
        style = MaterialTheme.typography.titleMedium
      )
    }
  )
}
@Composable
fun DownloadRoute(
  modifier: Modifier = Modifier,
  onDownloadComplete: (String, Path, ColorState) -> Unit,
  viewModel: DownloadViewModel = hiltViewModel()
) {
  val downloadUiState by viewModel.downloadUiState.collectAsStateWithLifecycle()

  DownloadScreen(
    modifier = modifier,
    downloadUiState = downloadUiState,
    onDownloadComplete = { path ->
      onDownloadComplete(viewModel.productId, path, viewModel.modelColor)
    }
  )
}
```

```
@Composable
private fun DownloadScreen(
    modifier: Modifier = Modifier,
    downloadUiState: DownloadUiState,
    onDownloadComplete: (Path) -> Unit
) {
    val onBackPressedDispatcher =
LocalOnBackPressedDispatcherOwner.current?.onBackPressedDispatcher
    var showQuitDialog by remember { mutableStateOf(false) }

    BackHandler(enabled = !showQuitDialog) {
        showQuitDialog = true
    }

    Box(
        modifier = modifier
            .fillMaxSize()
            .padding(LocalDimens.SpacingLarge),
    ) {
        Column(
            modifier = Modifier
                .fillMaxSize(),
            verticalArrangement = Arrangement.Center,
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Icon(
                modifier = Modifier.size(LocalDimens.IconXL),
                imageVector = Icons.Default.CloudDownload,
                contentDescription = null
            )

            Spacer(modifier = Modifier.height(LocalDimens.SpacingMedium))

            when (downloadUiState) {
                DownloadUiState.Loading -> InitialLoading()

                is DownloadUiState.Progress -> DownloadProgress(
                    progressState = downloadUiState
                )

                is DownloadUiState.Completed -> {
                    DownloadComplete(location = downloadUiState.path.toString())
                    onDownloadComplete(downloadUiState.path)
                }

                is DownloadUiState.Failed -> Timber.d(downloadUiState.reason, downloadUiState.e)
            }
        }

        if (showQuitDialog) {
            QuitDialog(
```

```
                message = stringResource(id = R.string.title_quit_download),
                onDismiss = { showQuitDialog = false }
            ) {
                showQuitDialog = false
                onBackPressedDispatcher?.onBackPressed()
            }
        }
    }
}

@Composable
private fun InitialLoading(
    modifier: Modifier = Modifier
) {
    CircularProgressIndicator(modifier = modifier)
}

@Composable
private fun DownloadProgress(
    modifier: Modifier = Modifier,
    progressState: DownloadUiState.Progress
) {

    val progress by animateFloatAsState(
        targetValue = progressState.progress,
        label = Constants.LABEL_DOWNLOAD_PROGRESS,
        animationSpec = tween(durationMillis = Constants.DOWNLOAD_PROGRESS_INTERVAL)
    )

    Column(
        modifier = modifier.fillMaxWidth(LocalDimens.Download.ProgressWidthPercentage),
        verticalArrangement = Arrangement.Center,
        horizontalAlignment = Alignment.CenterHorizontally
    ) {
        LinearProgressIndicator(
            modifier = Modifier
                .fillMaxWidth()
                .padding(vertical = LocalDimens.SpacingSmall),
            progress = { progress },
            strokeCap = StrokeCap.Round
        )

        Text(
            text = stringResource(
                id = R.string.downloading,
                progressState.progress.toPercentage().roundToInt()
            )
        )
    }
}

@Composable
```

```
private fun DownloadComplete(
    modifier: Modifier = Modifier,
    location: String
) {
    Text(
        modifier = modifier,
        text = stringResource(R.string.download_completed, location)
    )
}
sealed interface DownloadUiState {

    data object Loading : DownloadUiState

    data class Progress(
        val downloaded: Long,
        val total: Long
    ) : DownloadUiState

    data class Completed(val path: Path) : DownloadUiState

    data class Failed(val reason: String, val e: Exception? = null) : DownloadUiState
}


val DownloadUiState.Progress.progress: Float
    get() = downloaded.toFloat() / total.toFloat()

@HiltViewModel
class DownloadViewModel @Inject constructor(
    savedStateHandle: SavedStateHandle,
    private val getDownloadInfoUseCase: GetDownloadInfoUseCase,
    private val downloadModelUseCase: DownloadModelUseCase,
    fileHelper: FileHelper
) : ViewModel() {

    private val args = DownloadArgs(savedStateHandle)

    val productId = args.productId
    val modelColor = args.modelColor

    private val _downloadUiState =
MutableStateFlow<DownloadUiState>(DownloadUiState.Loading)
    val downloadUiState = _downloadUiState.asStateFlow()

    private val downloadDirectory = fileHelper.getModelDir()

    init {
        download(args.productId)
    }

    private fun download(productId: String) {
        // Get local path to store the downloaded model
```

```
    val path = downloadDirectory.resolve("$productId.glb")

    viewModelScope.launch {
        getDownloadInfoUseCase(productId).collectLatest { result ->
            when (result) {
                Result.Loading -> {
                    _downloadUiState.emit(DownloadUiState.Loading)
                }

                is Result.Success -> {
                    // Check if the model is already downloaded and has the same size
                    // as available in download info
                    if (path.checkIfExist(result.data.totalSize)) {
                        _downloadUiState.emit(DownloadUiState.Completed(path))
                        return@collectLatest
                    }
                    startDownload(path, result.data.url)
                }

                is Result.Error -> {
                    Timber.d("Failed to get product $productId details.")
                    // Use cached model if exist
                    if (path.checkIfExist()) {
                        Timber.d("Using cached model.")
                        _downloadUiState.emit(DownloadUiState.Completed(path))
                    }
                }
            }
        }
    }
}

private suspend fun startDownload(path: Path, url: String) {
    downloadModelUseCase(path, url).collectLatest { downloadStatus ->
        when (downloadStatus) {
            DownloadStatus.Loading -> {
                _downloadUiState.emit(DownloadUiState.Loading)
            }

            is DownloadStatus.Error -> {
                Timber.e(downloadStatus.message, downloadStatus.cause)
                _downloadUiState.emit(
                    DownloadUiState.Failed(
                        downloadStatus.message
                    )
                )
            }

            is DownloadStatus.Progress -> {
                _downloadUiState.emit(
                    DownloadUiState.Progress(
                        downloadStatus.downloaded,
```

```
                        downloadStatus.total
                    ).also { Timber.i("Progress: ${it.progress.toPercentage()}") }
                )
            }

            DownloadStatus.Completed -> {
                _downloadUiState.emit(
                    DownloadUiState.Completed(path)
                )
            }
        }
    }
}

@Composable
fun Modifier.shimmerEffect(
    color: Color = MaterialTheme.colorScheme.secondaryContainer
): Modifier = composed {

    var size by remember {
        mutableStateOf(IntSize.Zero)
    }

    // Make animation infinite
    val transition = rememberInfiniteTransition(label = "Shimmer")

    // Offset representing starting position for the effect
    val startOffsetX by transition.animateFloat(
        initialValue = -2 * size.width.toFloat(),
        targetValue = 2 * size.width.toFloat(),
        animationSpec = infiniteRepeatable(
            animation = tween(
                durationMillis = LocalDimens.ShimmerEffect.AnimationDuration
            )
        ),
        label = "Shimmer"
    )

    background(
        brush = Brush.linearGradient(
            colors = listOf(
                color.copy(alpha = LocalDimens.ShimmerEffect.ColorAlpha),
                color,
                color.copy(alpha = LocalDimens.ShimmerEffect.ColorAlpha),
                color
            ),
            start = Offset(startOffsetX, 0f),
            end = Offset(startOffsetX + size.width.toFloat(), size.height.toFloat())
        ),
        shape = CardDefaults.shape
    )
```

```
        .onGloballyPositioned {
            // Set measured size for use in effect.
            size = it.size
        }
}

const val PRODUCTS_ROUTE = "products_route"

fun NavController.navigateToProducts() {
    navigate(PRODUCTS_ROUTE)
}

fun NavGraphBuilder.productsScreen(onProductClick: (String) -> Unit) {
    composable(
        route = PRODUCTS_ROUTE
    ) {
        ProductsRoute(onProductClick = onProductClick)
    }
}

@Composable
fun ProductsRoute(
    modifier: Modifier = Modifier,
    onProductClick: (String) -> Unit,
    viewModel: ProductsViewModel = hiltViewModel()
) {
    val currentCategory by viewModel.currentCategory.collectAsStateWithLifecycle()
    val productItems = viewModel.modelsFlow.collectAsLazyPagingItems()

    ProductsScreen(
        modifier = modifier,
        categories = viewModel.categories,
        currentCategory = currentCategory,
        productItems = productItems,
        onCategoryChange = viewModel::changeCategory,
        onProductClick = onProductClick
    )
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun ProductsScreen(
    modifier: Modifier = Modifier,
    categories: List<Category>,
    currentCategory: Category,
    productItems: LazyPagingItems<Product>,
    onCategoryChange: (Category) -> Unit,
    onProductClick: (String) -> Unit
) {

    val pullToRefreshState = rememberPullToRefreshState()
```

```
    if (pullToRefreshState.isRefreshing) {
        LaunchedEffect(Unit) {
            productItems.refresh()
            pullToRefreshState.endRefresh()
        }
    }

    Box(
        modifier = modifier
            .fillMaxSize()
            .nestedScroll(pullToRefreshState.nestedScrollConnection)
    ) {
        Column {
            CategoryHeader(
                categories = categories,
                currentCategory = currentCategory,
                onCategoryChange = onCategoryChange
            )

            Products(
                productItems = productItems,
                onProductClick = onProductClick
            )
        }

        if (productItems.loadState.refresh !is LoadState.Loading) {
            PullToRefreshContainer(
                modifier = Modifier.align(Alignment.TopCenter),
                state = pullToRefreshState
            )
        }
    }
}

@Composable
private fun CategoryHeader(
    modifier: Modifier = Modifier,
    categories: List<Category>,
    currentCategory: Category,
    onCategoryChange: (Category) -> Unit
) {
    LazyRow(
        modifier = modifier
            .padding(top = LocalDimens.SpacingSmall, bottom = LocalDimens.SpacingXS),
        contentPadding = PaddingValues(horizontal = LocalDimens.SpacingLarge),
        horizontalArrangement = Arrangement.spacedBy(LocalDimens.SpacingSmall)
    ) {
        items(categories) { category ->
            CategoryCard(
                isSelected = currentCategory == category,
                category = category,
                onClick = { onCategoryChange(category) }
```

```
            )
        }
    }
}

@Composable
private fun CategoryCard(
    modifier: Modifier = Modifier,
    category: Category,
    isSelected: Boolean,
    onClick: () -> Unit
) {
    FilterChip(
        selected = isSelected,
        modifier = modifier,
        onClick = onClick,
        label = {
            Text(text = stringResource(category.nameRes))
        }
    )
}

@Composable
fun Products(
    modifier: Modifier = Modifier,
    productItems: LazyPagingItems<Product>,
    onProductClick: (String) -> Unit
) {
    Box(modifier = modifier.fillMaxSize()) {
        val refreshState = productItems.loadState.refresh
        val sourceState = productItems.loadState.source.refresh
        val mediatorState = productItems.loadState.mediator?.refresh

        when {
            // If there is no data available to display, show no data view.
            // Initially item counts might be zero. We must check that source
            // state or remote mediator is not in loading state.
            productItems.itemCount == 0
                    && sourceState !is LoadState.Loading
                    && mediatorState !is LoadState.Loading -> {
                NoData(modifier = Modifier.align(Alignment.Center))
            }

            // When products are loading show product placeholders.
            // Loading can be either by refresh state (initial load/pull to refresh) or
            // by sourceState when changing the category.
            refreshState is LoadState.Loading
                    || (sourceState is LoadState.Loading && productItems.itemCount == 0) -> {
                ProductsLoading()
            }

            else -> {
```

```
            ProductsContent(
                products = productItems,
                onProductClick = onProductClick
            )
        }
    }
}

@Composable
private fun ProductsLoading(
    modifier: Modifier = Modifier
) {
    ProductsGrid(modifier = modifier) {
        items(LocalDimens.Products.PlaceholdersCount) {
            Box(
                modifier = modifier
                    .aspectRatio(LocalDimens.Products.CardRatio)
                    .shimmerEffect()
            )
        }
    }
}

@Composable
private fun ProductsContent(
    modifier: Modifier = Modifier,
    products: LazyPagingItems<Product>,
    onProductClick: (String) -> Unit
) {
    ProductsGrid(modifier = modifier) {
        items(products.itemCount) { index ->
            products[index]?.let {
                ProductCard(
                    model = it,
                    onProductClick = onProductClick
                )
            }
        }
    }
}

@Composable
private fun ProductsGrid(
    modifier: Modifier = Modifier,
    content: LazyGridScope.() -> Unit
) {
    LazyVerticalGrid(
        modifier = modifier
            .padding(horizontal = LocalDimens.SpacingLarge),
        contentPadding = PaddingValues(vertical = LocalDimens.SpacingMedium),
        columns = GridCells.Adaptive(LocalDimens.Products.CardSize),
```

```
        verticalArrangement = Arrangement.spacedBy(LocalDimens.SpacingLarge),
        horizontalArrangement = Arrangement.spacedBy(LocalDimens.SpacingMedium),
        content = content
    )
}

@Composable
private fun ProductCard(
    modifier: Modifier = Modifier,
    model: Product,
    onProductClick: (String) -> Unit
) {
    Card(
        modifier = modifier
            .aspectRatio(ratio = LocalDimens.Products.CardRatio),
        onClick = { onProductClick(model.id) }
    ) {
        SubcomposeAsyncImage(
            modifier = Modifier.weight(1f),
            model = model.thumbnail,
            contentDescription = model.name,
            contentScale = ContentScale.FillHeight,
            loading = {
                Icon(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(LocalDimens.SpacingSmall),
                    painter = painterResource(id = R.drawable.ic_model_placeholder),
                    contentDescription = null,
                    tint = LocalContentColor.current.copy(alpha =
LocalDimens.Products.PlaceholderAlpha)
                )
            },
            error = {
                Icon(
                    modifier = Modifier
                        .fillMaxSize()
                        .padding(LocalDimens.SpacingSmall),
                    imageVector = Icons.Default.BrokenImage,
                    contentDescription = null,
                    tint = LocalContentColor.current.copy(alpha =
LocalDimens.Products.PlaceholderAlpha)
                )
            }
        )

        Text(
            modifier = Modifier.padding(LocalDimens.SpacingSmall),
            text = model.name,
            overflow = TextOverflow.Ellipsis,
            maxLines = 2
        )
```

```
    }
}

@Composable
private fun NoData(
    modifier: Modifier = Modifier
) {
    Surface(
        modifier = modifier,
        contentColor = MaterialTheme.colorScheme.onSecondaryContainer
    ) {
        Column(
            verticalArrangement = Arrangement.spacedBy(LocalDimens.SpacingSmall)
        ) {
            Icon(
                modifier = modifier.size(LocalDimens.IconXXL),
                imageVector = Icons.AutoMirrored.Default.List,
                contentDescription = null
            )

            Text(
                text = stringResource(R.string.no_data),
                style = MaterialTheme.typography.headlineMedium
            )
        }
    }
}
@HiltViewModel
class ProductsViewModel @Inject constructor(
    private val getProductsUseCase: GetProductsUseCase,
    getProductCategoriesUseCase: GetProductCategoriesUseCase
) : ViewModel() {

    val categories = getProductCategoriesUseCase()

    private val _currentCategory = MutableStateFlow(Category.TABLE)
    val currentCategory = _currentCategory.asStateFlow()


    @OptIn(ExperimentalCoroutinesApi::class)
    val modelsFlow = currentCategory.flatMapLatest { categories ->
        Timber.d("Updating category: $categories")
        getProductsUseCase(categories)
    }

    fun changeCategory(category: Category) {
        viewModelScope.launch {
            _currentCategory.update { category }
        }
    }
}
```

```
val md_theme_light_primary = Color(0xFF984717)
val md_theme_light_onPrimary = Color(0xFFFFFFFF)
val md_theme_light_primaryContainer = Color(0xFFFFDBCB)
val md_theme_light_onPrimaryContainer = Color(0xFF341100)
val md_theme_light_secondary = Color(0xFF765849)
val md_theme_light_onSecondary = Color(0xFFFFFFFF)
val md_theme_light_secondaryContainer = Color(0xFFFFDBCB)
val md_theme_light_onSecondaryContainer = Color(0xFF2C160B)
val md_theme_light_tertiary = Color(0xFF655F31)
val md_theme_light_onTertiary = Color(0xFFFFFFFF)
val md_theme_light_tertiaryContainer = Color(0xFFECE4AA)
val md_theme_light_onTertiaryContainer = Color(0xFF1F1C00)
val md_theme_light_error = Color(0xFFBA1A1A)
val md_theme_light_errorContainer = Color(0xFFFFDAD6)
val md_theme_light_onError = Color(0xFFFFFFFF)
val md_theme_light_onErrorContainer = Color(0xFF410002)
val md_theme_light_background = Color(0xFFFFFBFF)
val md_theme_light_onBackground = Color(0xFF201A18)
val md_theme_light_surface = Color(0xFFFFFBFF)
val md_theme_light_onSurface = Color(0xFF201A18)
val md_theme_light_surfaceVariant = Color(0xFFF4DED5)
val md_theme_light_onSurfaceVariant = Color(0xFF52443D)
val md_theme_light_outline = Color(0xFF85736C)
val md_theme_light_inverseOnSurface = Color(0xFFFBEEE9)
val md_theme_light_inverseSurface = Color(0xFF362F2C)
val md_theme_light_inversePrimary = Color(0xFFFFB692)
val md_theme_light_shadow = Color(0xFF000000)
val md_theme_light_surfaceTint = Color(0xFF984717)
val md_theme_light_outlineVariant = Color(0xFFD7C2B9)
val md_theme_light_scrim = Color(0xFF000000)

val md_theme_dark_primary = Color(0xFFFFB692)
val md_theme_dark_onPrimary = Color(0xFF552000)
val md_theme_dark_primaryContainer = Color(0xFF793100)
val md_theme_dark_onPrimaryContainer = Color(0xFFFFDBCB)
val md_theme_dark_secondary = Color(0xFFE6BEAC)
val md_theme_dark_onSecondary = Color(0xFF432A1E)
val md_theme_dark_secondaryContainer = Color(0xFF5C4033)
val md_theme_dark_onSecondaryContainer = Color(0xFFFFDBCB)
val md_theme_dark_tertiary = Color(0xFFD0C890)
val md_theme_dark_onTertiary = Color(0xFF353107)
val md_theme_dark_tertiaryContainer = Color(0xFF4C481C)
val md_theme_dark_onTertiaryContainer = Color(0xFFECE4AA)
val md_theme_dark_error = Color(0xFFFFB4AB)
val md_theme_dark_errorContainer = Color(0xFF93000A)
val md_theme_dark_onError = Color(0xFF690005)
val md_theme_dark_onErrorContainer = Color(0xFFFFDAD6)
val md_theme_dark_background = Color(0xFF201A18)
val md_theme_dark_onBackground = Color(0xFFEDE0DB)
val md_theme_dark_surface = Color(0xFF201A18)
val md_theme_dark_onSurface = Color(0xFFEDE0DB)
val md_theme_dark_surfaceVariant = Color(0xFF52443D)
```

```
val md_theme_dark_onSurfaceVariant = Color(0xFFD7C2B9)
val md_theme_dark_outline = Color(0xFFA08D85)
val md_theme_dark_inverseOnSurface = Color(0xFF201A18)
val md_theme_dark_inverseSurface = Color(0xFFEDE0DB)
val md_theme_dark_inversePrimary = Color(0xFF984717)
val md_theme_dark_shadow = Color(0xFF000000)
val md_theme_dark_surfaceTint = Color(0xFFFFB692)
val md_theme_dark_outlineVariant = Color(0xFF52443D)
val md_theme_dark_scrim = Color(0xFF000000)

val seed = Color(0xFFC46A39)


@Composable
fun AppNavHost(
    modifier: Modifier = Modifier,
    appState: AppState,
    startDestination: String = PRODUCTS_ROUTE,
    onShowSnackbar: suspend (String, String?) -> Boolean
) {

    val navController = appState.navController

    NavHost(
        navController = navController,
        startDestination = startDestination,
        modifier = modifier,
    ) {
        productsScreen { productId ->
            Timber.d("Clicked: $productId")
            navController.navigateToDownload(productId)
        }

        downloadScreen { productId, _, color ->
            navController.navigateToARView(productId, color)
        }

        arViewScreen(
            onNavigateBack = { navController.navigateToProducts() },
            onShowSnackbar = onShowSnackbar
        )
    }
}


fun ARSceneView.captureImage(): Bitmap {
    // Create a bitmap the size of the scene view.
    val bitmap = Bitmap.createBitmap(
        width, height,
        Bitmap.Config.ARGB_8888
    )
```

```
    // Create a handler thread to offload the processing of the image.
    val handlerThread = HandlerThread(Constants.IMAGE_CAPTURE_HANDLER_NAME)
    handlerThread.start()

    PixelCopy.request(this, bitmap, { copyResult ->
        if (copyResult == PixelCopy.SUCCESS) {
            Timber.d("Created bitmap from ARSceneView success.")
        } else {
            Timber.e("Failed to create bitmap from ARSceneView.")
        }
        handlerThread.quitSafely()
    }, Handler(handlerThread.looper))

    return bitmap
}

class YuvToRgbConverter(context: Context) {
    private val rs = RenderScript.create(context)
    private val scriptYuvToRgb = ScriptIntrinsicYuvToRGB.create(rs, Element.U8_4(rs))

    private var pixelCount: Int = -1
    private lateinit var yuvBuffer: ByteBuffer
    private lateinit var inputAllocation: Allocation
    private lateinit var outputAllocation: Allocation

    @Synchronized
    fun yuvToRgb(image: Image, output: Bitmap) {

        // Ensure that the intermediate output byte buffer is allocated
        if (!::yuvBuffer.isInitialized) {
            pixelCount = image.cropRect.width() * image.cropRect.height()
            yuvBuffer = ByteBuffer.allocateDirect(
                pixelCount * ImageFormat.getBitsPerPixel(ImageFormat.YUV_420_888) / 8
            )
        }

        // Get the YUV data in byte array form
        imageToByteBuffer(image, yuvBuffer)

        // Ensure that the RenderScript inputs and outputs are allocated
        if (!::inputAllocation.isInitialized) {
            inputAllocation = Allocation.createSized(rs, Element.U8(rs), yuvBuffer.array().size)
        }
        if (!::outputAllocation.isInitialized) {
            outputAllocation = Allocation.createFromBitmap(rs, output)
        }

        // Convert YUV to RGB
        inputAllocation.copyFrom(yuvBuffer.array())
        scriptYuvToRgb.setInput(inputAllocation)
        scriptYuvToRgb.forEach(outputAllocation)
        outputAllocation.copyTo(output)
```

```
}

private fun imageToByteBuffer(image: Image, outputBuffer: ByteBuffer) {
    assert(image.format == ImageFormat.YUV_420_888)

    val imageCrop = image.cropRect
    val imagePlanes = image.planes
    val rowData = ByteArray(imagePlanes.first().rowStride)

    imagePlanes.forEachIndexed { planeIndex, plane ->

        // How many values are read in input for each output value written
        // Only the Y plane has a value for every pixel, U and V have half the resolution i.e.
        //
        // Y Plane          U Plane    V Plane
        // ===============  =======    =======
        // Y Y Y Y Y Y Y Y   U U U U    V V V V
        // Y Y Y Y Y Y Y Y   U U U U    V V V V
        // Y Y Y Y Y Y Y Y   U U U U    V V V V
        // Y Y Y Y Y Y Y Y   U U U U    V V V V
        // Y Y Y Y Y Y Y Y
        // Y Y Y Y Y Y Y Y
        // Y Y Y Y Y Y Y Y
        val outputStride: Int

        // The index in the output buffer the next value will be written at
        // For Y it's zero, for U and V we start at the end of Y and interleave them i.e.
        //
        // First chunk        Second chunk
        // ===============    ===============
        // Y Y Y Y Y Y Y Y    U V U V U V U V
        // Y Y Y Y Y Y Y Y    U V U V U V U V
        // Y Y Y Y Y Y Y Y    U V U V U V U V
        // Y Y Y Y Y Y Y Y    U V U V U V U V
        // Y Y Y Y Y Y Y Y
        // Y Y Y Y Y Y Y Y
        // Y Y Y Y Y Y Y Y
        var outputOffset: Int

        when (planeIndex) {
            0 -> {
                outputStride = 1
                outputOffset = 0
            }

            1 -> {
                outputStride = 2
                outputOffset = pixelCount + 1
            }

            2 -> {
                outputStride = 2
```

```
                outputOffset = pixelCount
            }

            else -> {
                // Image contains more than 3 planes, something strange is going on
                return@forEachIndexed
            }
        }

        val buffer = plane.buffer
        val rowStride = plane.rowStride
        val pixelStride = plane.pixelStride

        // We have to divide the width and height by two if it's not the Y plane
        val planeCrop = if (planeIndex == 0) {
            imageCrop
        } else {
            Rect(
                imageCrop.left / 2,
                imageCrop.top / 2,
                imageCrop.right / 2,
                imageCrop.bottom / 2
            )
        }

        val planeWidth = planeCrop.width()
        val planeHeight = planeCrop.height()

        buffer.position(rowStride * planeCrop.top + pixelStride * planeCrop.left)
        for (row in 0 until planeHeight) {
            val length: Int
            if (pixelStride == 1 && outputStride == 1) {
                // When there is a single stride value for pixel and output, we can just copy
                // the entire row in a single step
                length = planeWidth
                buffer.get(outputBuffer.array(), outputOffset, length)
                outputOffset += length
            } else {
                // When either pixel or output have a stride > 1 we must copy pixel by pixel
                length = (planeWidth - 1) * pixelStride + 1
                buffer.get(rowData, 0, length)
                for (col in 0 until planeWidth) {
                    outputBuffer.array()[outputOffset] = rowData[col * pixelStride]
                    outputOffset += outputStride
                }
            }

            if (row < planeHeight - 1) {
                buffer.position(buffer.position() + rowStride - length)
            }
        }
    }
```

```
    }
}

@Composable
fun rememberYuvToRgbConverter(): YuvToRgbConverter {
    val context = LocalContext.current
    return remember {
        YuvToRgbConverter(context)
    }
}



private const val HTTP_LOGGING_INTERCEPTOR = "HTTP_LOGGING_INTERCEPTOR"
private const val HTTP_DOWNLOAD_LOGGING_INTERCEPTOR =
"HTTP_DOWNLOAD_LOGGING_INTERCEPTOR"
private const val OKHTTP_CLIENT = "OKHTTP_CLIENT"
const val OKHTTP_DOWNLOAD_CLIENT = "OKHTTP_DOWNLOAD_CLIENT"
private const val CACHE_SIZE: Long = 10 * 1024 * 1024 // 10 MB
private const val NETWORK_CALL_TIMEOUT: Long = 1 // Minute

@Module
@InstallIn(SingletonComponent::class)
object ApiModule {

    @Singleton
    @Provides
    @Named(HTTP_LOGGING_INTERCEPTOR)
    fun providesHttpLoggingInterceptor(): HttpLoggingInterceptor = HttpLoggingInterceptor().apply
{
        level = if (BuildConfig.DEBUG)
            HttpLoggingInterceptor.Level.BODY
        else
            HttpLoggingInterceptor.Level.NONE
    }

    @Singleton
    @Provides
    @Named(HTTP_DOWNLOAD_LOGGING_INTERCEPTOR)
    fun providesHttpDownloadLoggingInterceptor(): HttpLoggingInterceptor =
        HttpLoggingInterceptor().apply {
            level = if (BuildConfig.DEBUG)
            // Get only Headers for debugging. Logging Body will download whole response (file body)
            // at once and progress won't work.
                HttpLoggingInterceptor.Level.HEADERS
            else
                HttpLoggingInterceptor.Level.NONE
        }

    @Singleton
    @Provides
    fun providesApiAuthenticator(): Authenticator = ApiAuthenticator()
```

```
    @Singleton
    @Provides
    @Named(OKHTTP_CLIENT)
    fun providesOkHttpClient(
        @ApplicationContext context: Context,
        @Named(HTTP_LOGGING_INTERCEPTOR) httpLoggingInterceptor:
HttpLoggingInterceptor,
        authenticator: Authenticator
    ): OkHttpClient = OkHttpClient.Builder()
        .cache(Cache(context.cacheDir, CACHE_SIZE))
        .addInterceptor(httpLoggingInterceptor)
        .authenticator(authenticator)
        .readTimeout(NETWORK_CALL_TIMEOUT, TimeUnit.MINUTES)
        .writeTimeout(NETWORK_CALL_TIMEOUT, TimeUnit.MINUTES)
        .build()

    @Singleton
    @Provides
    @Named(OKHTTP_DOWNLOAD_CLIENT)
    fun providesOkHttpDownloadClient(
        @Named(HTTP_DOWNLOAD_LOGGING_INTERCEPTOR) httpLoggingInterceptor:
HttpLoggingInterceptor
    ): OkHttpClient = OkHttpClient.Builder()
        .addInterceptor(httpLoggingInterceptor)
        .readTimeout(NETWORK_CALL_TIMEOUT, TimeUnit.MINUTES)
        .writeTimeout(NETWORK_CALL_TIMEOUT, TimeUnit.MINUTES)
        .build()

    @Singleton
    @Provides
    fun providesRetrofit(
        @Named(OKHTTP_CLIENT) okHttpClient: OkHttpClient,
        apiResultCallAdapterFactory: ApiResultCallAdapterFactory
    ): Retrofit = Retrofit.Builder()
        .baseUrl(
            Urls.getBaseUrl()
                .also { Timber.d("Setting Base URL : $it") }
        )
        .client(okHttpClient)
        .addConverterFactory(GsonConverterFactory.create())
        .addCallAdapterFactory(apiResultCallAdapterFactory)
        .build()

    @Singleton
    @Provides
    fun providesApiService(
        retrofit: Retrofit
    ): ApiService = retrofit.create(ApiService::class.java)
}
```

```
@Entity(
    tableName = Constants.TABLE_CATEGORY_PRODUCT,
    primaryKeys = ["categoryId", "productId"],
    foreignKeys = [
        ForeignKey(
            entity = CategoryEntity::class,
            parentColumns = ["id"],
            childColumns = ["categoryId"],
            onDelete = ForeignKey.CASCADE
        ),
        ForeignKey(
            entity = ProductEntity::class,
            parentColumns = ["id"],
            childColumns = ["productId"],
            onDelete = ForeignKey.CASCADE
        )
    ]
)
data class CategoryProductCrossRef(
    @ColumnInfo(index = true)
    val categoryId: Long,
    @ColumnInfo(index = true)
    val productId: String
)


@Entity(
    tableName = Constants.TABLE_CATEGORY,
    indices = [
        Index(value = ["category"], unique = true)
    ]
)
data class CategoryEntity(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    val category: Category,
    val planeType: PlaneType
)

fun CategoryInfo.asEntity() = CategoryEntity(category = category, planeType = planeType)

@Database(
    entities = [
        ProductEntity::class,
        RemoteKey::class,
        CategoryEntity::class,
        CategoryProductCrossRef::class
    ],
    version = 1,
    exportSchema = false
)
@TypeConverters(Converters::class)
```

```
abstract class SSFurniCraftARDatabase : RoomDatabase() {

    abstract fun modelDao(): ProductDao

    abstract fun remoteKeyDao(): RemoteKeyDao

    abstract fun categoryAndProductDao(): CategoryAndProductDao
}
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА
МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ВИКОРИСТАННЯ ДОПОВНЕНОЇ РЕАЛЬНОСТІ В МОДЕРНІЗОВАНИХ АНДРОЇД ЗАСТОСУНКАХ

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

Магістерська кваліфікаційна робота на тему:
«Методи та програмні засоби використання доповненої реальності в модернізованих андроїд застосунках »

Автор: ст.групи 1ПІ-23м Карабіньвоський Д.М,
Науковий керівник: к.т.н., доцент каф. ПЗ Черноволик Г.О.

Вінниця – 2024

Рисунок Г.1 – Титульний слайд

Актуальність теми

Швидкий розвиток сучасних технологій суттєво змінює способи обслуговування клієнтів і підходи до підвищення конкурентоспроможності бізнесу. Це особливо актуально для сфери розробки мобільних додатків, де конкуренція є високою, а вимоги користувачів до якості продуктів постійно зростають. Традиційні методи взаємодії з користувачами, які ще нещодавно задовольняли основні потреби аудиторії, вже не відповідають сучасним стандартам і очікуванням. Сьогодні користувачі шукають більше ніж просто функціональність – вони хочуть отримувати якісний цифровий досвід, що включає інноваційні підходи, інтерактивність та персоналізацію.

Однією з технологій, яка активно змінює підходи до створення мобільних додатків, є доповнена реальність (AR). Вона дозволяє інтегрувати в цифрове середовище візуальні, інтерактивні елементи, які роблять процес взаємодії з додатком значно цікавішим і зручнішим для користувачів. Наприклад, AR може використовуватися для демонстрації товарів у тривимірному форматі, створення віртуальних турів, інтерактивного навчання чи навіть розважальних функцій. Такі технології дозволяють не лише розширити функціональні можливості додатків, але й покращити загальний досвід користувачів, створюючи позитивні асоціації та підвищуючи лояльність до бренду.

Завдяки таким інноваціям мобільні додатки стають не просто засобом для виконання функцій, а повноцінним інструментом для взаємодії з користувачем, який залишає позитивні враження. Тому актуальною є розробка нових методів і підходів до інтеграції технології доповненої реальності в мобільні додатки, що дозволить не лише задовольнити сучасні потреби користувачів, а й створити унікальні рішення, які забезпечать бізнесу конкурентну перевагу.

2

Рисунок Г.2 – Актуальність теми

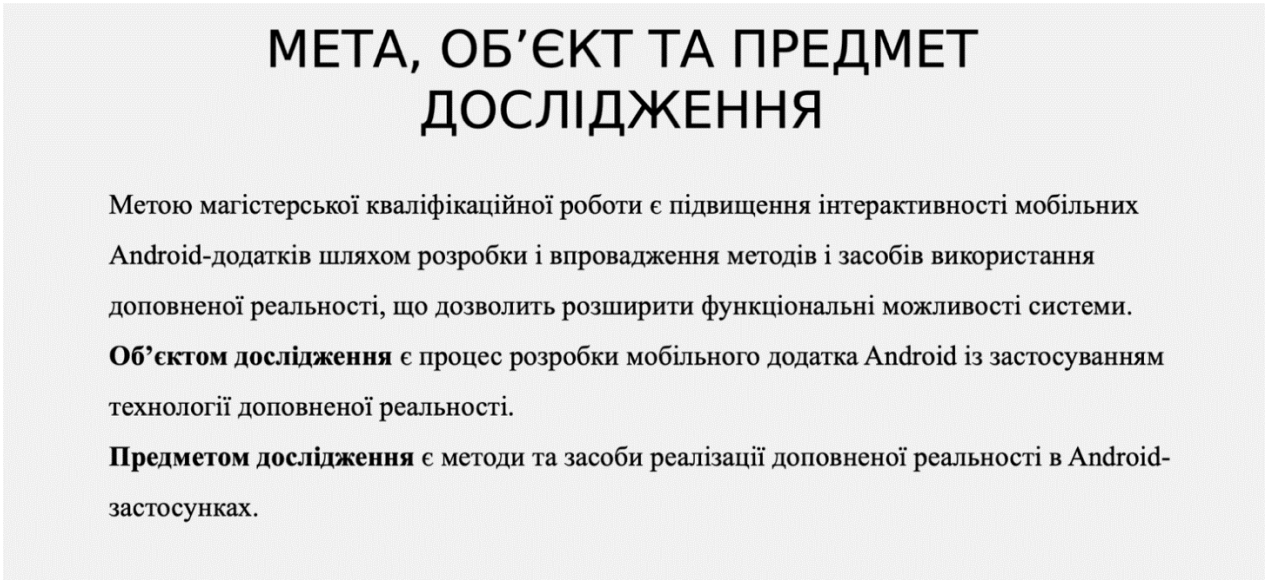

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

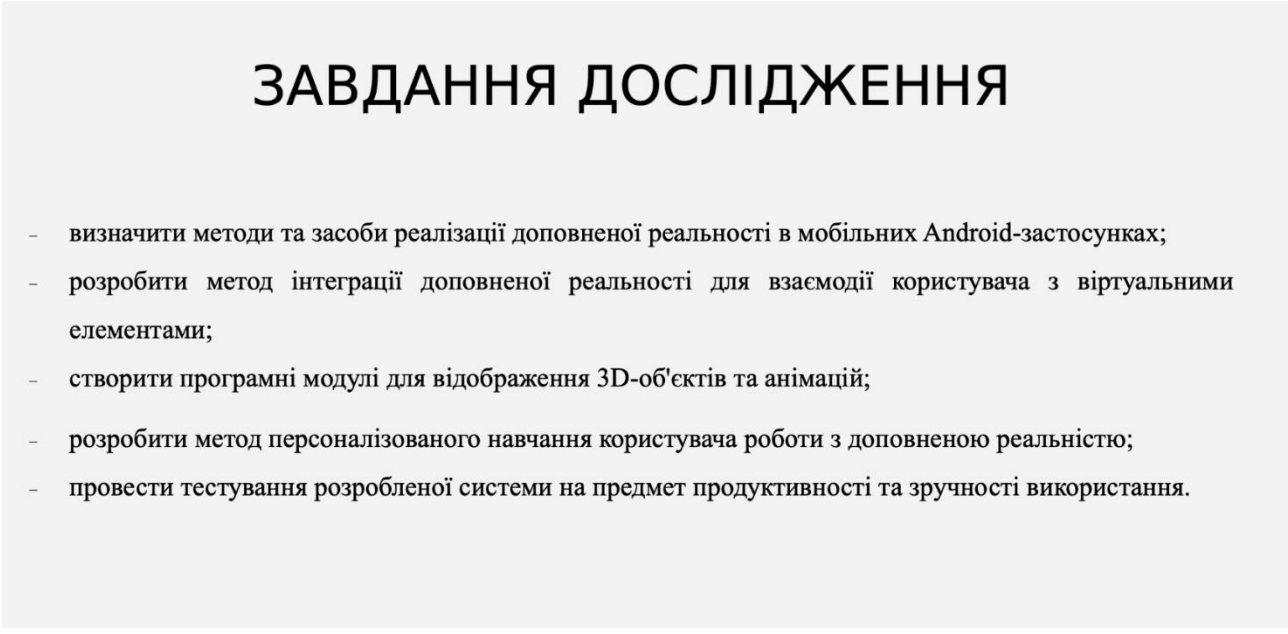

Рисунок Г.4 – Завдання дослідження

НАУКОВА НОВИЗНА

1. Подальший розвиток отримав метод інтерактивного відображення даних і взаємодії з користувачем за допомогою доповненої реальності, який, на відміну від існуючих, використовує анімований оверлей, орієнтований на швидке реагування на дії користувача, що сприяє покращенню користувацького досвіду і поліпшить взаємодію користувача з середовищем мобільної системи;
2. Подальшого розвитку отримав метод створення віртуальних елементів та 3D-об'єктів, який, на відміну від існуючих, автоматизує процеси інтеграції об'єктів у віртуальний простір за допомогою додаткового кешування характеристик об'єкту, що дозволяє розширити функціонал Android системи в процесі взаємодії з об'єктами доповненої реальності.

Рисунок Г.5 – Наукова новизна

ПРАКТИЧНА ЦІННІСТЬ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Розроблена Android-система з функціями доповненої реальності може застосовуватися для покращення користувацького досвіду в різних сферах, таких як освіта, роздрібна торгівля, туризм, надаючи інтерактивний контент у режимі реального часу.

Рисунок Г.6 – Практична цінність отриманих результатів

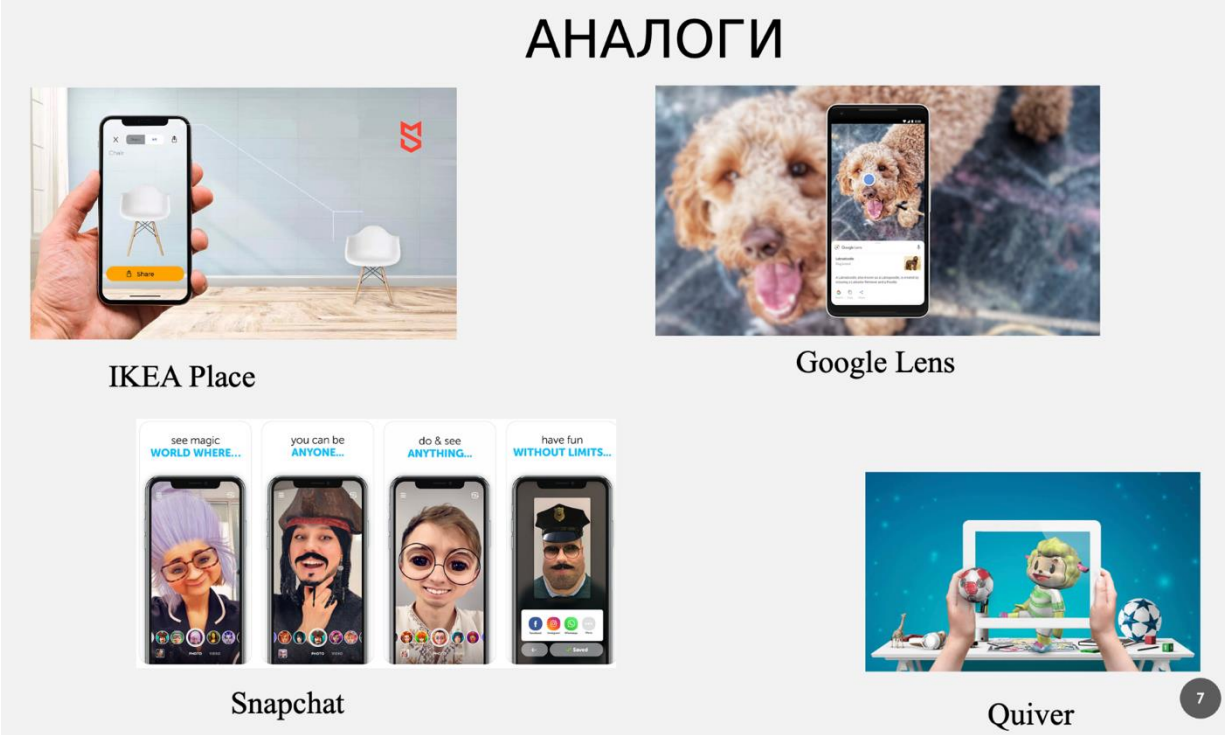

Рисунок Г.7 – Аналоги

ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ

Назва додатку / Критерій	IKEA Place	Google Lens	Snapchat	Quiver	AccuVein	AR Room
Реалістичність і точність AR-контенту	+	+	-	-	+	+
Наявність персоналізованого ознайомлення	-	-	-	-	-	+
Адаптивність до навколишнього середовища	+	+	+	-	-	+
Сумісність з пристроями	-	+	+	+	-	+
Продуктивність	+	-	-	-	+	+
Підсумковий результат	3	2	2	2	2	5

Рисунок Г.8 – Порівняльний аналіз аналогів

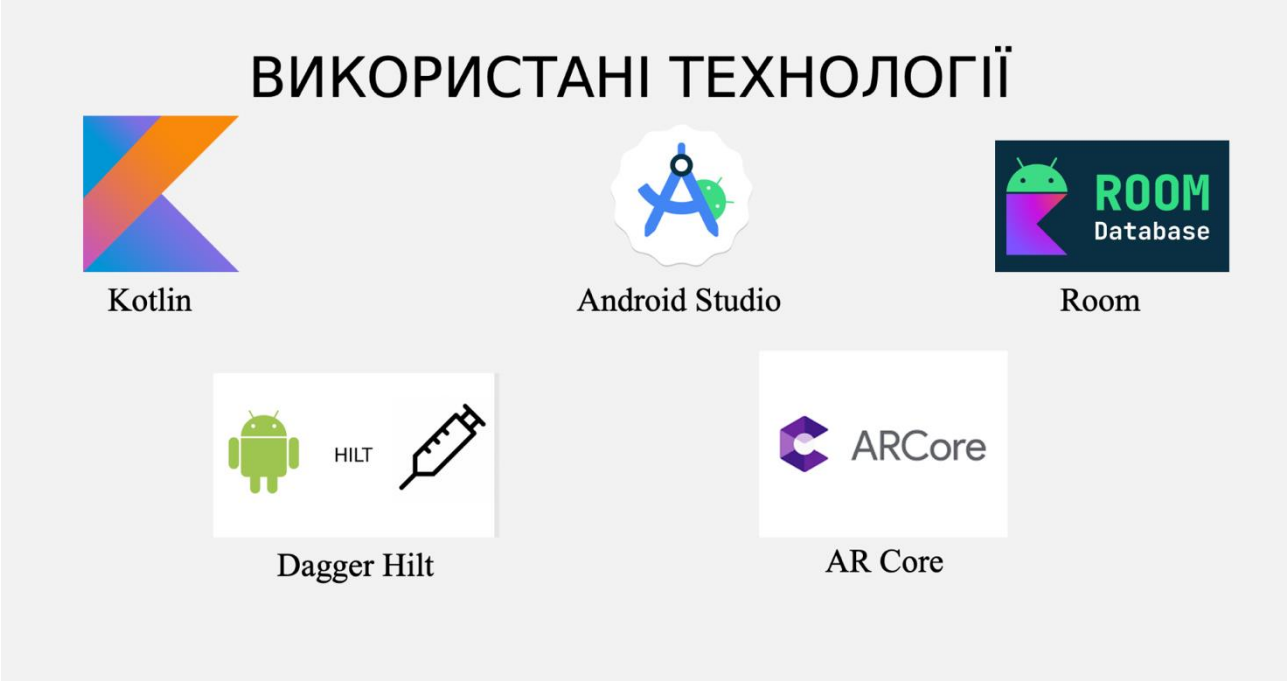

Рисунок Г.9 – Використані технології

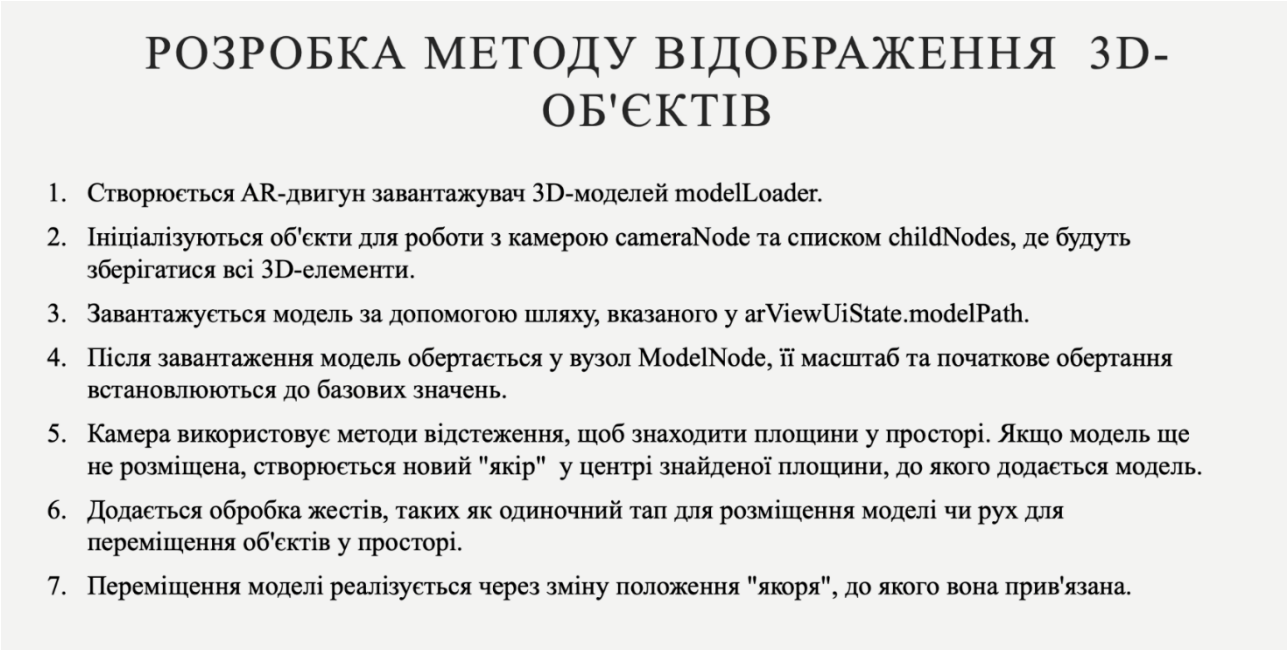

Рисунок Г.10 – Розробка методу відображення 3D-об'єктів

РОЗРОБКА МЕТОДУ ВІДОБРАЖЕННЯ 3D-ОБ'ЄКТІВ

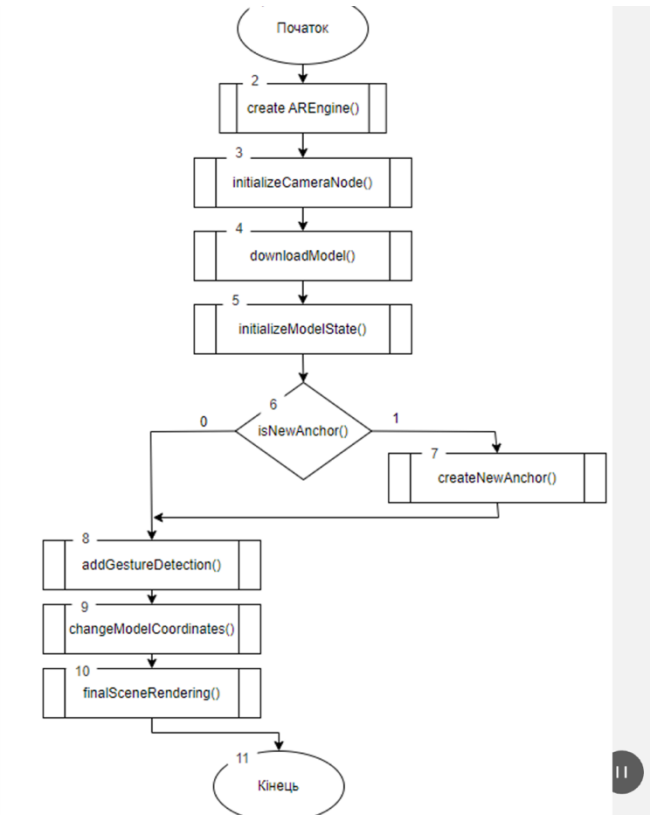

Рисунок Г.11 – Алгоритм методу відображення 3D-об'єктів

РОЗРОБКА МЕТОДУ ПЕРСОНАЛІЗОВАНОГО НАВЧАННЯ КОРИСТУВАЧА РОБОТИ З ДОПОВНЕНОЮ РЕАЛЬНІСТЮ

1) задавання анімації жестів через LottieCompositionSpec::RawRes;
2) визначення відповідних текстових повідомлення для кожного жесту через stringResource;
3) використовується змінна playedCount, яка зберігається з можливістю відновлення стану через rememberSaveable;
4) розраховування індексу поточного жесту (currentGestureIndex) залежно від кількості програних анімацій;
5) якщо кількість ітерацій для кожного жесту перевищує задану кількість, викликається метод onComplete;
6) використовується rememberLottieComposition для зберігання стану анімації;
7) коли прогрес анімації досягає кінця ANIM_COMPLETE_VALUE, викликається функція onEnd, яка збільшує playedCount;
8) анімації повторюються для кожного жесту до досягнення потрібної кількості ітерацій.

Рисунок Г.12 – Розробка методу персоналізованого навчання користувача роботи з доповненою реальністю

РОЗРОБКА МЕТОДУ ПЕРСОНАЛІЗОВАНОГО НАВЧАННЯ КОРИСТУВАЧА РОБОТИ З ДОПОВНЕНОЮ РЕАЛЬНІСТЮ

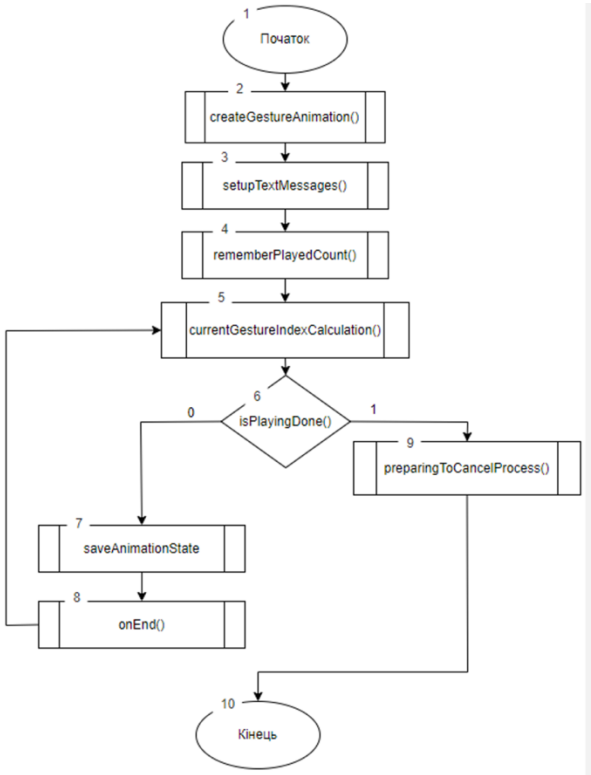

Рисунок Г.13 – Алгоритм методу персоналізованого навчання користувача роботи з доповненою реальністю

РОЗРОБКА МОДЕЛІ ANDROID - СИСТЕМИ

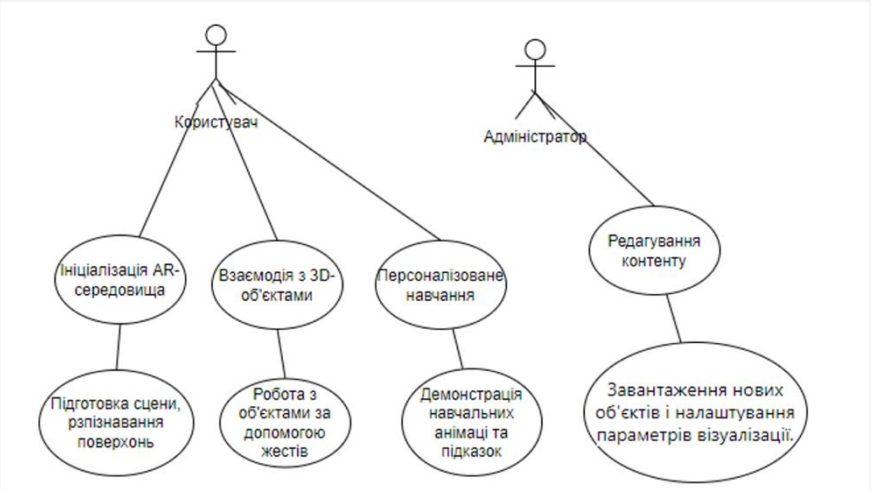

Рисунок Г.14 – Модель Android - системи

Рисунок Г.15 – Модель системи

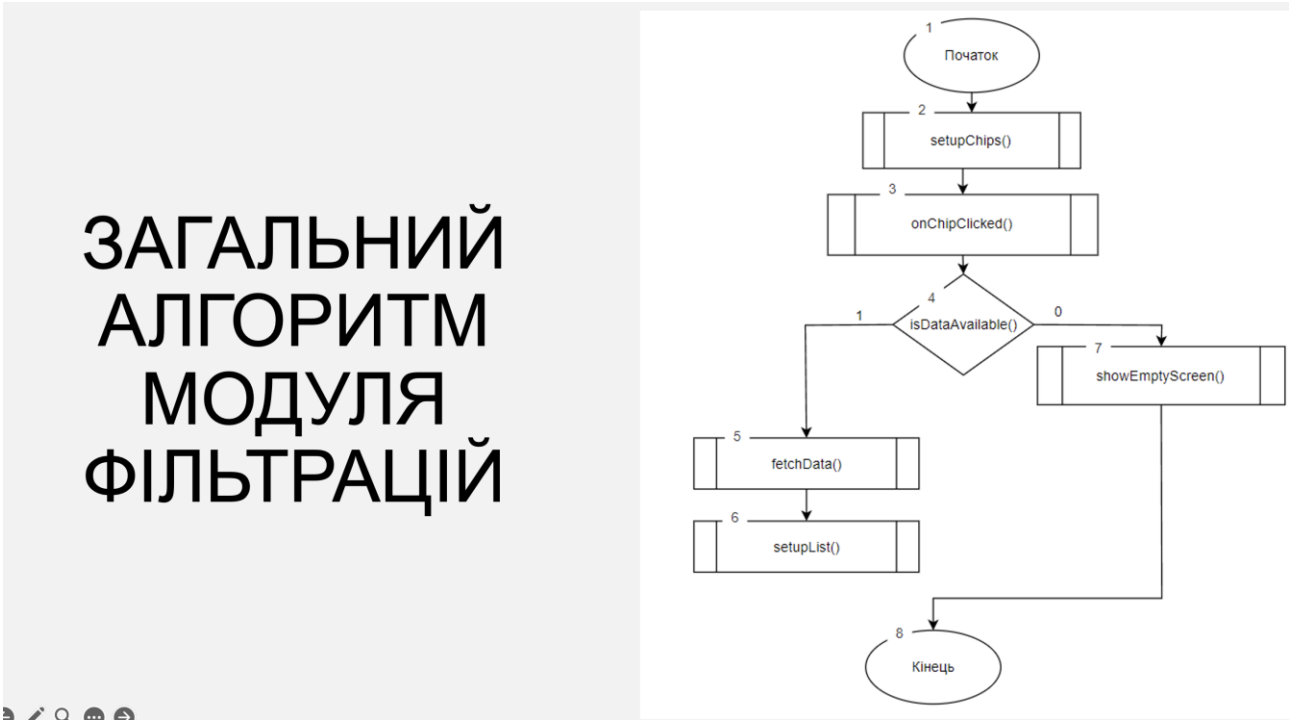

Рисунок Г.16 – Алгоритм модуля фільтрацій

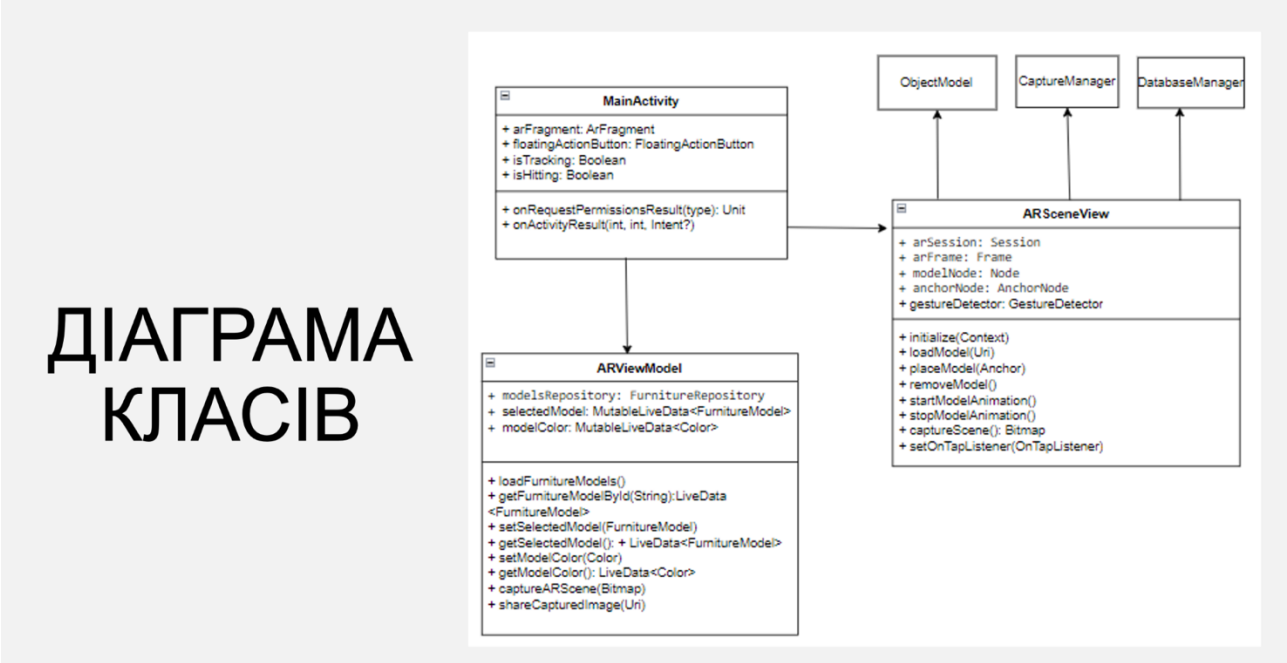

Рисунок Г.17 – Діаграма класів

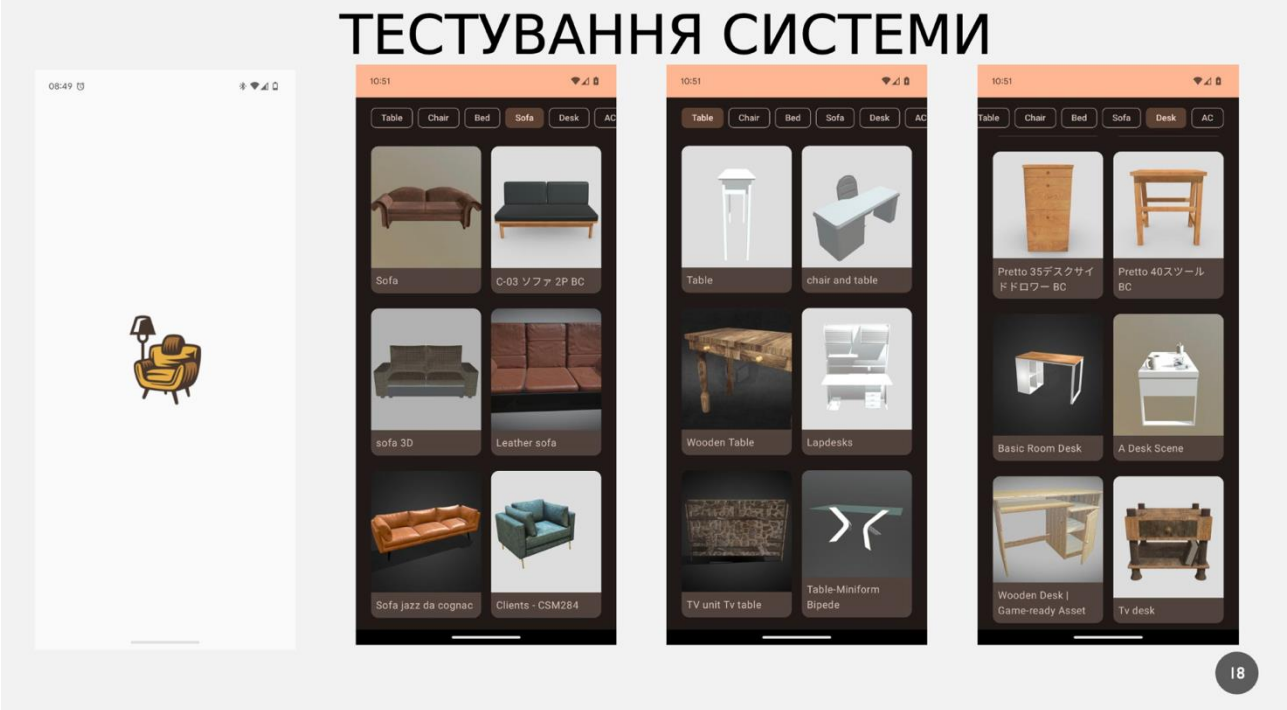

Рисунок Г.18 – Тестування системи (частина 1)

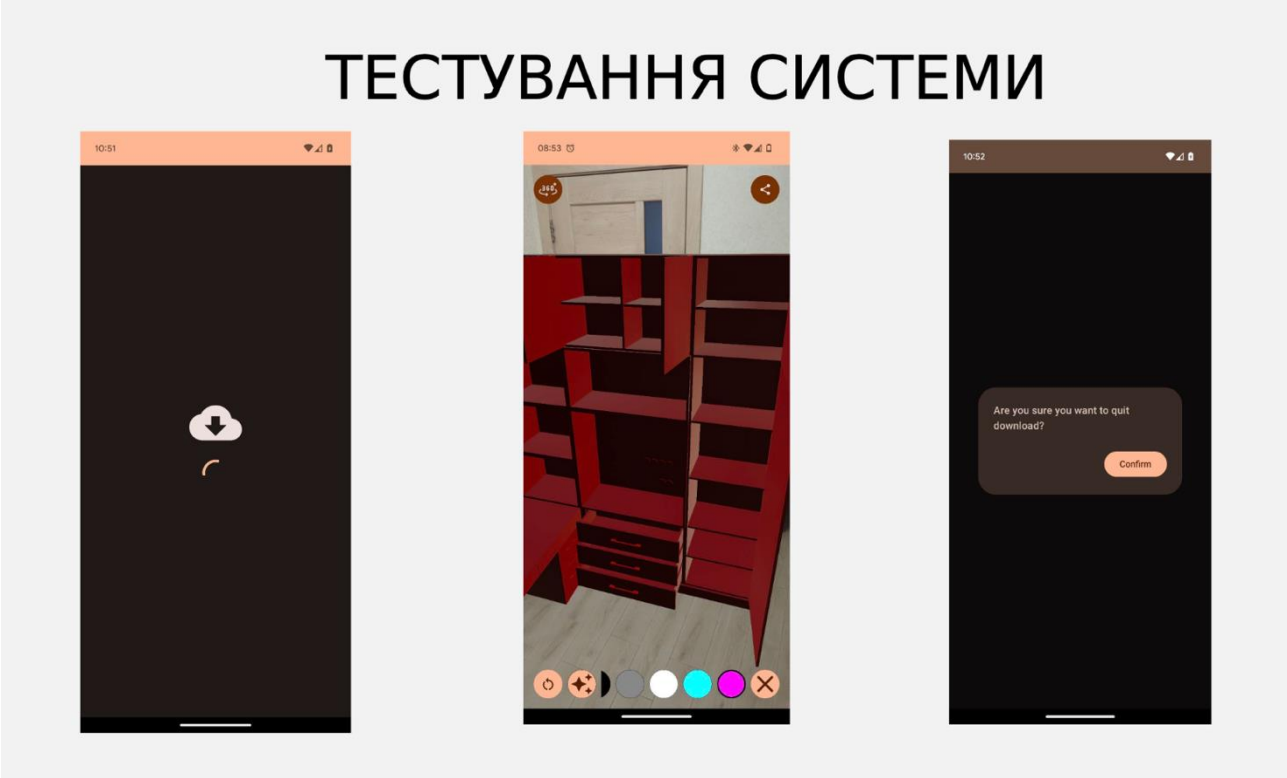

Рисунок Г.19 – Тестування системи (частина 2)

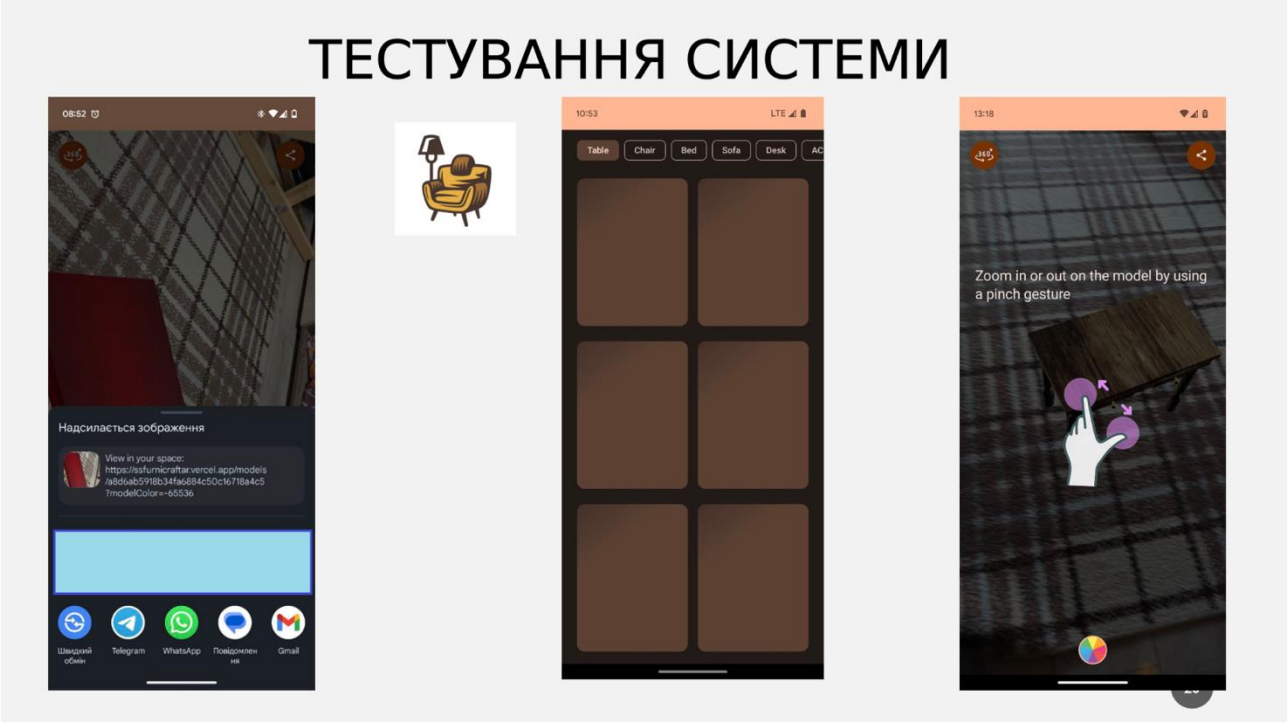

Рисунок Г.20 – Тестування системи (частина 3)

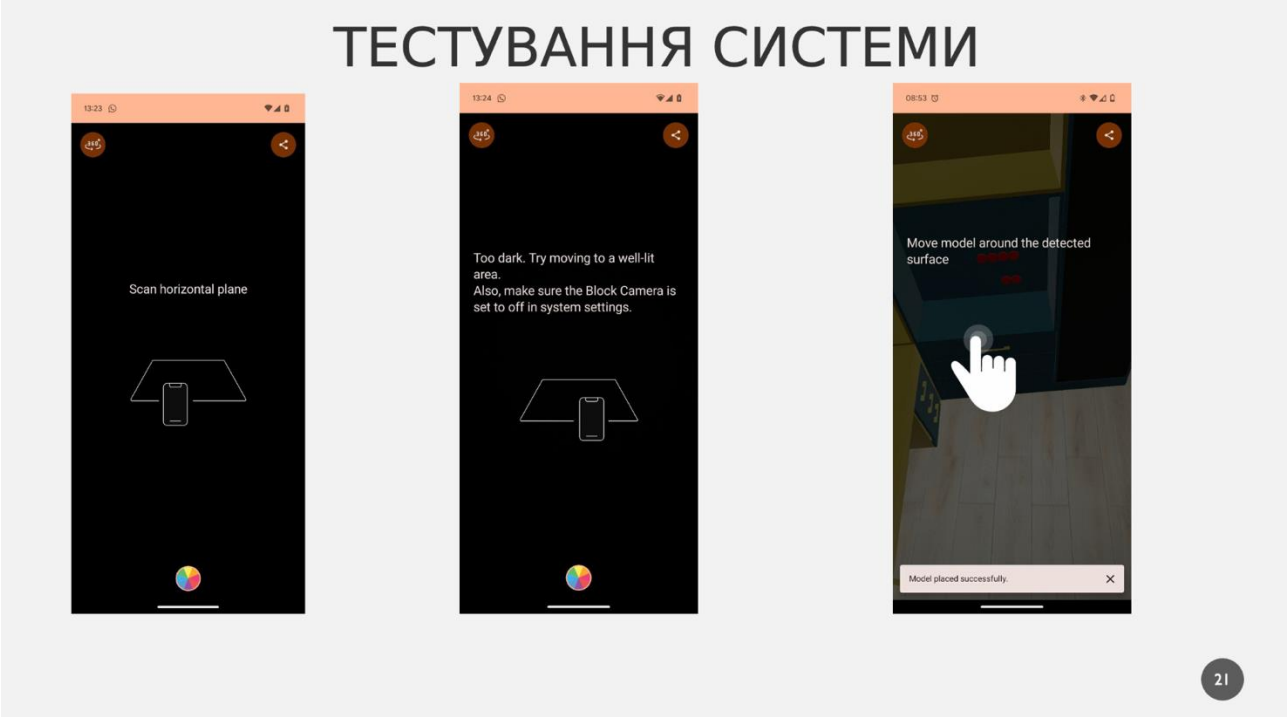

Рисунок Г.21 – Тестування системи (частина 4)

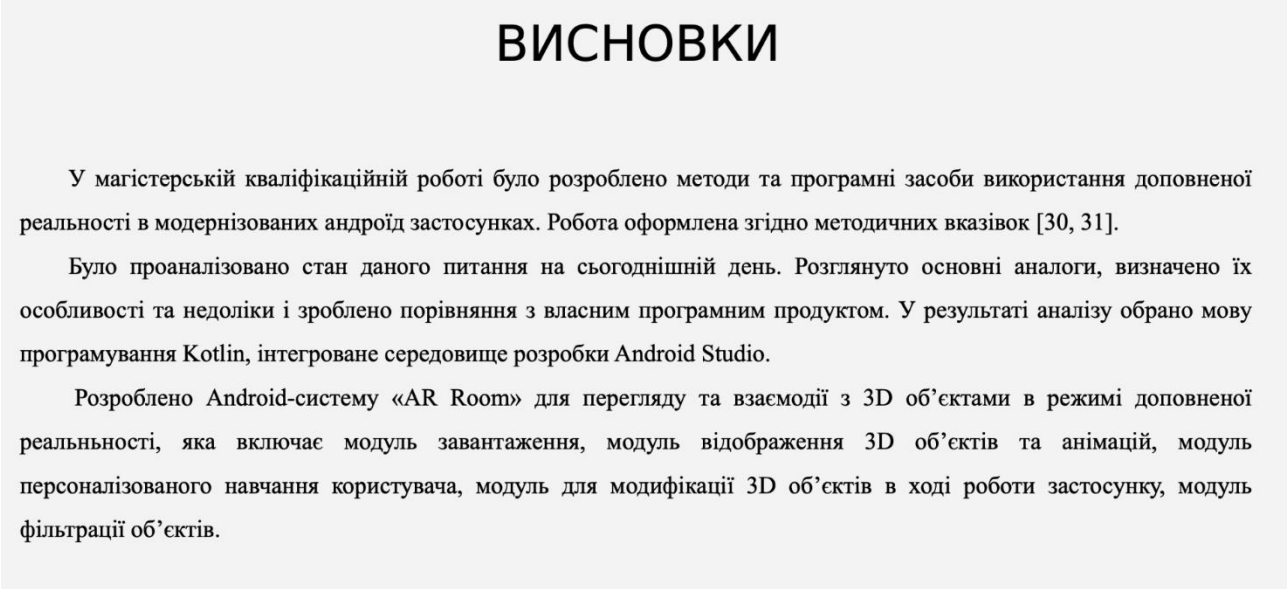

Рисунок Г.22 – Висновки (частина 1)

ВИСНОВКИ

Покращено метод інтерактивного відображення даних і взаємодії з користувачем за допомогою доповненої реальності, який використовує анімований оверлей в додатку, що орієнтований на швидке реагування на дії користувача, та покращить користувацький досвід від додатку;

Подальшого розвитку отримав метод створення віртуальних елементів та 3D-об'єктів, який, автоматизує процеси інтеграції об'єктів у віртуальний простір за допомогою додаткового кешування характеристик об'єкту, що вплине на швидкодію додатку та дозволить користуватись ним навіть на девайсах з низькими характеристиками.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Розроблено інструкцію користувача.

Рисунок Г.23 – Висновки (частина 2)

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ І ПУБЛІКАЦІЇ

Апробація матеріалів. Результати магістерської кваліфікаційної роботи обговорювалися на XVII міжнародній науково-практичній конференції «Інформаційні технології і автоматизація – 2024» (Одеса, 2024).

Публікації. Войтко В.В. Застосування AR для персоналізованих покупок і вибору товарів в android-додатках / В. В. Войтко, Д.М. Карабіньовський, А.В.Денисюк // Інформаційні технології і автоматизація – 2024 / Матеріали XVII міжнародної науково-практичної конференції. Одеса, 31 жовтня – 1 листопада 2024 р. – Одеса, Видавництво ОНТУ, 2024 р. – С. 451-453.

Рисунок Г.24 – Апробації та публікації

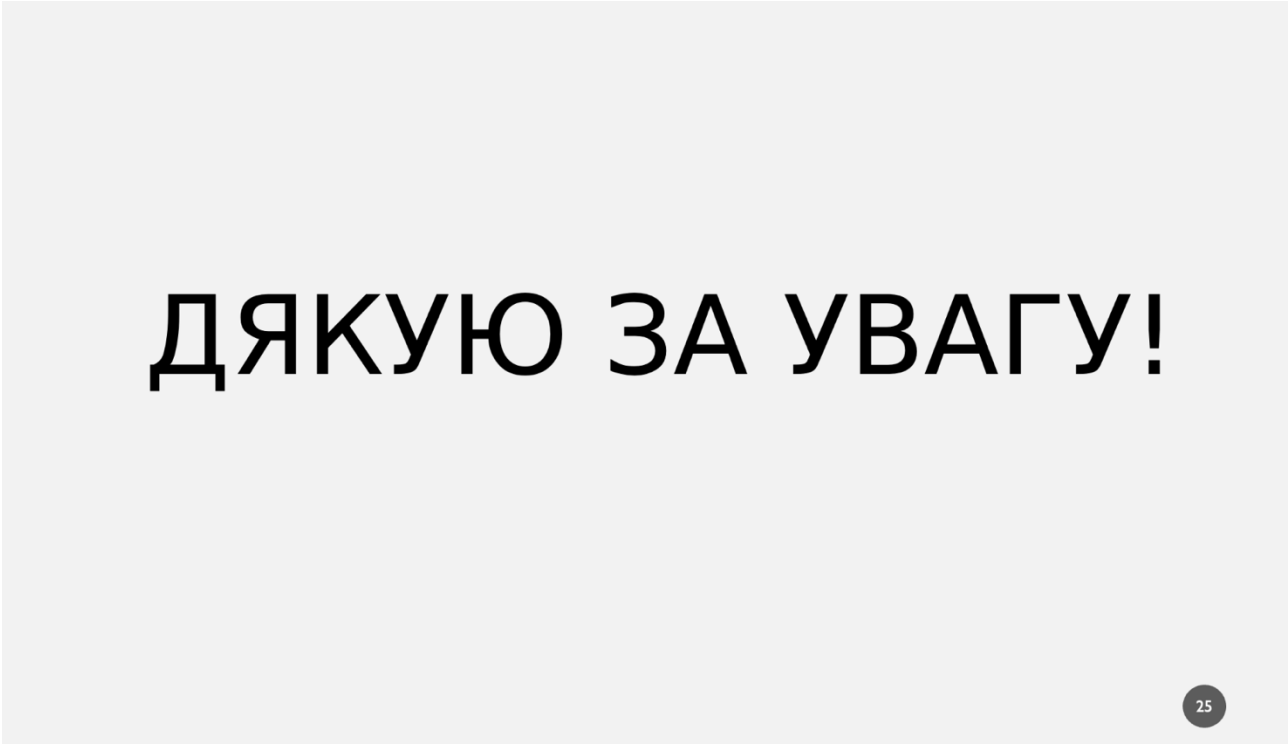

Рисунок Г.25 – Кінцевий слайд