Міністерство освіти і науки України
Криворізький національний університет
Кафедра комп’ютерних систем та мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
за спеціальністю 123 «Комп’ютерна інженерія»

Тема наукової роботи: ПРОГРАМНІ ЗАСОБИ ТА МОДЕЛІ ДЛЯ РЕАЛІЗАЦІЇ РНОТОМАТН НА ПЛАТФОРМІ ANDROID

Виконав _____ А. А. Макаревич

Керівник _____ С. В. Сьомочкина

Консультант _____ Д. І. Кузнєцов

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2023

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти магістр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

« » _____ 2023 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Макаревича Андрія Анатолійовича
(прізвище, ім'я, по батькові)

1. Тема роботи Програмні засоби та моделі для реалізації Photomath на платформі Android

керівник роботи Сьомочкина Світлана Володимирівна, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "__" ____ 2023 року №_
2. Строк подання студентом роботи_____
3. Вихідні дані до роботи Операційна система – Android

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Зміст, вступ, аналіз засобів та технологій створення Android-додатків, розробка мобільного додатка Photomath, алгоритм роботи мобільного додатка, реалізація методів основного функціоналу, тестування, висновок, список використаних джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Презентація (12 слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1.	Початок написання кваліфікаційної роботи	04. 09. 2023	
2.	Збір теоретичного матеріалу	04. 09. 2023 – 31. 10. 2023	
3.	Вибір методів реалізації додатка та написання першого розділу	01. 11. 2023 – 07. 11. 2023	
4.	Реалізація методів основного функціоналу додатка	08. 11. 2023 – 15. 11. 2023	
5.	Написання другого розділу	16. 11. 2023 – 26. 11. 2023	
6.	Написання третього розділу	27. 11. 2023 – 28. 11. 2023	
7.	Оформлення кваліфікаційної роботи	29. 11. 2023 – 02. 12. 2023	

Студент_____ _____
(підпис) (прізвище та ініціали)

Керівник роботи_____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 90 сторінок, 32 рисунки, 15 лістингів, 3 таблиці, 20 використаних джерел.

Об’єкт проєктування – додаток Photomath на Android.

Проєкт складається з трьох розділів.

Перший розділ присвячений детальному огляду операційної системи Android, проведено огляд віртуальних машин Android, архітектурних моделей для розроблення мобільних прикладних програм, засобів розробки мобільних прикладних програм та існуючих рішень на задану тему. Зроблено постановку завдання та вибрано методи реалізації додатка.

В другому розділі було розглянуто алгоритм розробки мобільного додатка та реалізовано основні методи для роботи додатка та створено основні Activity.

Третій розділ присвячений тестуванню загальної функціональності додатка та тестуванню сумісності.

МОБІЛЬНИЙ ДОДАТОК, ANDROID, JAVA, PHOTOMATH

Explanatory note: 90 pages, 32 figures, 15 listings, 3 tables, 20 used sources.

Project object – Photomath application on Android.

The project consists of three sections.

The first chapter is devoted to a detailed overview of the Android operating system, an overview of Android virtual machines, architectural models for mobile application development, mobile application development tools, and existing solutions on the given topic was conducted. The task was formulated and the application implementation methods were selected.

In the second section, the mobile application development algorithm was considered and the main methods for the application were implemented and the main Activities were created.

The third section is devoted to testing the general functionality of the application and testing compatibility.

MOBILE APP, ANDROID, JAVA, PHOTOMATH

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ОС – Операційна система;
API – Application Programming Interface;
APK – Android Package;
ART – Android Runtime;
C2DM – Cloud to Device Messaging;
GPS – Global Positioning System;
JVM – Java Virtual Machine;
MVC – Model View Controller;
NFC – Near-Field Communication;
OHA – Open Handset Alliance;
OMAP – Open Multimedia Application Platform;
SDK – Software Development Kit;
VM – Virtual Machine.

ВСТУП

В сучасному світі, де мобільні технології стали невід’ємною частиною нашого повсякденного життя, однією з актуальних проблем є розробка програмних засобів для реалізації завдань, пов'язаних з обробкою і аналізом зображень, зокрема в галузі освіти. Одним із важливих завдань цієї сфери є розробка програм для автоматичного розпізнавання і розв'язання математичних задач на основі фотографій. У цьому контексті Photomath, додаток для смартфонів, що здатний розпізнавати і вирішувати математичні завдання, виконує важливу функцію.

Аналізуючи вітчизняну та зарубіжну науково-технічну літературу та патентний пошук, можна визначити, що функціональність Photomath великою мірою базується на комплексі алгоритмів машинного навчання, розпізнаванні зображень та розумових обчисленнях. Водночас, існують певні технічні та методологічні проблеми, пов'язані з розширенням функціоналу додатка та його оптимізацією для платформи Android.

Актуальність роботи. Актуальність даної роботи полягає у вдосконаленні функціональності та доступності Photomath для користувачів Android. З урахуванням широкого поширення цієї операційної системи, важливо створити оптимальні програмні засоби та моделі для ефективної реалізації Photomath на даній платформі. Такий крок сприятиме покращенню якості освіти та розширенню можливостей користувачів у вивченні математики. Актуальністю також є пошук нових методів обробки та аналізу зображень, що дозволить покращити точність та швидкість вирішення математичних завдань з використанням фотографій.

Мета і завдання дослідження. Метою даної дипломної роботи є розробка програмних засобів та моделей для реалізації функціоналу Photomath на платформі Android. Дана робота ґрунтується на ідеї розвитку та оптимізації програм для освіти та навчання з використанням передових методів машинного навчання та розпізнавання зображень. Основні наукові положення цієї роботи полягатимуть у розробці та реалізації ефективних алгоритмів для розпізнавання математичних завдань на фотографіях, а також у вивченні оптимальних підходів до їх оптимізації для платформи Android.

Для досягнення цієї мети необхідно виконати наступні завдання:

1. Вивчити та проаналізувати існуючі програмні засоби та моделі для розв'язання математичних задач.
2. Розробити алгоритм розв'язання математичних задач, який буде використовуватись в програмі. Розробити інтерфейс користувача та взаємодію з камерою пристрою для отримання вхідних даних.

3. Провести тестування програми та внести необхідні корективи для поліпшення її функціональності та надійності.

Об’єкт дослідження – процеси автоматичного розпізнавання математичних завдань на фотографіях.

Предмет дослідження – методи та технології, які можна використовувати для оптимальної адаптації Photomath під Android, включаючи алгоритми обробки зображень, оптимізацію роботи програми та розширення функціоналу.

Методи досліджень. У роботі буде використано аналіз існуючих рішень, програмування на мові Java для розробки Android-додатків, тестування ефективності та надійності реалізації.

Наукова новизна одержаних результатів. Вивчено актуальні концепції та підходи до створення додатків для розпізнавання та вирішення математичних завдань, що дозволило розробити інноваційний алгоритм для ефективного вирішення цих завдань. Алгоритм, розроблений на основі останніх досягнень у галузі, відрізняється кількома ключовими аспектами, що покращують його ефективність та користувацький досвід. По-перше, реалізовано передовий механізм автоматичного розпізнавання математичних виразів з фотографій. Цей механізм використовує сучасні алгоритми комп'ютерного зору та нейронних мереж для точного визначення складних математичних структур та оптимізації їхнього розпізнавання. Це дозволяє користувачам отримувати надійні та точні результати в найбільш заплутаних математичних виразах. По-друге, розроблений алгоритм включає інтелектуальні стратегії розв'язання різноманітних математичних задач. Ці стратегії базуються на аналізі структури завдань та використовують оптимізовані методи розв'язання для забезпечення швидкості та точності вирішення. По-третє, алгоритм взаємодіє з користувачем у інтуїтивно зрозумілому інтерфейсі, що дозволяє легко використовувати всі функції додатку. Це робить процес введення та вирішення математичних завдань максимально зручним для користувача.

Практичне значення отриманих результатів. Створено мобільний додаток Photomath для платформи Android, який включає в себе реалізацію основного функціоналу (налаштування камери, розпізнавання тексту та створення калькулятора); проведено тестування працездатності додатку Photomath.

Апробація роботи. Апробація досліджень відбувалась на конференції КІСМ-2023 від Криворізького національного університету.

1 АНАЛІЗ ЗАСОБІВ ТА ТЕХНОЛОГІЙ СТВОРЕННЯ ANDROID-ДОДАТКІВ

1.1 Основні відомості про Android

Android, як операційна система для мобільних пристроїв, завдяки своїй надзвичайно цікавій історії розвитку, визначає сучасний ландшафт мобільного ринку. Перша гучна згадка про Android з'явилася в 2005 році, коли Google придбав маленький стартап-проект Android, Inc. Ця дія викликала безліч спекуляцій та питань щодо майбутніх намірів Google на мобільному ринку.

Кульмінацією цієї еволюції став реліз Android 1.0 у 2008 році, який позначив початок нової ери для мобільних операційних систем. Android стала новим гравцем на перспективному ринку мобільних пристроїв, запустивши справжню битву з вже усталеними платформами, такими як iOS (раніше відомий як iPhone OS) та Blackberry.

Однією з ключових особливостей Android є те, що ця операційна система базується на відкритому вихідному коді, що робить бар'єр для входження для виробників пристроїв досить низьким. Вони можуть випускати пристрої для різних цінових сегментів та модифікувати систему для оптимальної взаємодії з конкретними пристроями. Таким чином, область застосування Android не обмежується лише смартфонами вищого цінового рівня, що робить її потенційну аудиторію досить широкою [1].

Ключовою складовою успіху Android стала освіта Open Handset Alliance (OHA) в кінці 2007 року. В OHA входять великі компанії, такі як HTC, Qualcomm, Motorola і NVIDIA, які спільно працюють над розробкою відкритих стандартів для мобільних пристроїв. Незважаючи на те, що ядро Android головним чином розробляється Google, всі члени OHA мають можливість внести свій внесок у проект у різних формах. Android, в свою чергу, представляє собою мобільну операційну систему і платформу, засновану на ядрі Linux версії 2.6, що вільна для використання у комерційних і некомерційних цілях. Багато членів OHA створюють власні версії Android для своїх пристроїв з власним користувацьким інтерфейсом, такими як HTC Sense або Motorola Motoblur.

Гнучкість та розширюваність Android завжди були важливими аспектами цієї мобільної операційної системи, проте ці переваги супроводжуються певними викликами та наслідками як для кінцевих користувачів, так і розробників додатків. Ця проблема, відома як «поділ Android», стала звичайним явищем в мобільному світі.

Для кінцевого користувача «поділ» Android призводить до обмежень у встановленні та використанні певних програм і можливостей, особливо для тих, хто залишається на старих версіях операційної системи. Процес оновлення для користувачів може бути незручним, іноді навіть неможливим через відсутність підтримки від виробників [1].

Для розробників «поділ» виражається в необхідності підтримувати роботу своїх додатків для різних версій Android. Додатки для старих версій зазвичай працюють на нових пристроях, але навпаки це не завжди правда. Деякі нові функції та можливості, які додаються в нових релізах Android, можуть бути недоступні в старих версіях. Це створює складнощі для програмістів, які змушені розділяти свій код та розробляти альтернативні варіанти для різних версій ОС.

Поділ Android, хоча і представляє виклик для розробників та користувачів, водночас є результатом бажання робити Android більш гнучким і сприятливим для різних виробників та різних категорій користувачів. Проте ефективне управління цим «поділом» залишається однією з важливих завдань для розробників та екосистеми Android, спрямованих на забезпечення оптимального досвіду користувачів та підтримку розробників у створенні додатків для цієї різноманітної операційної системи.

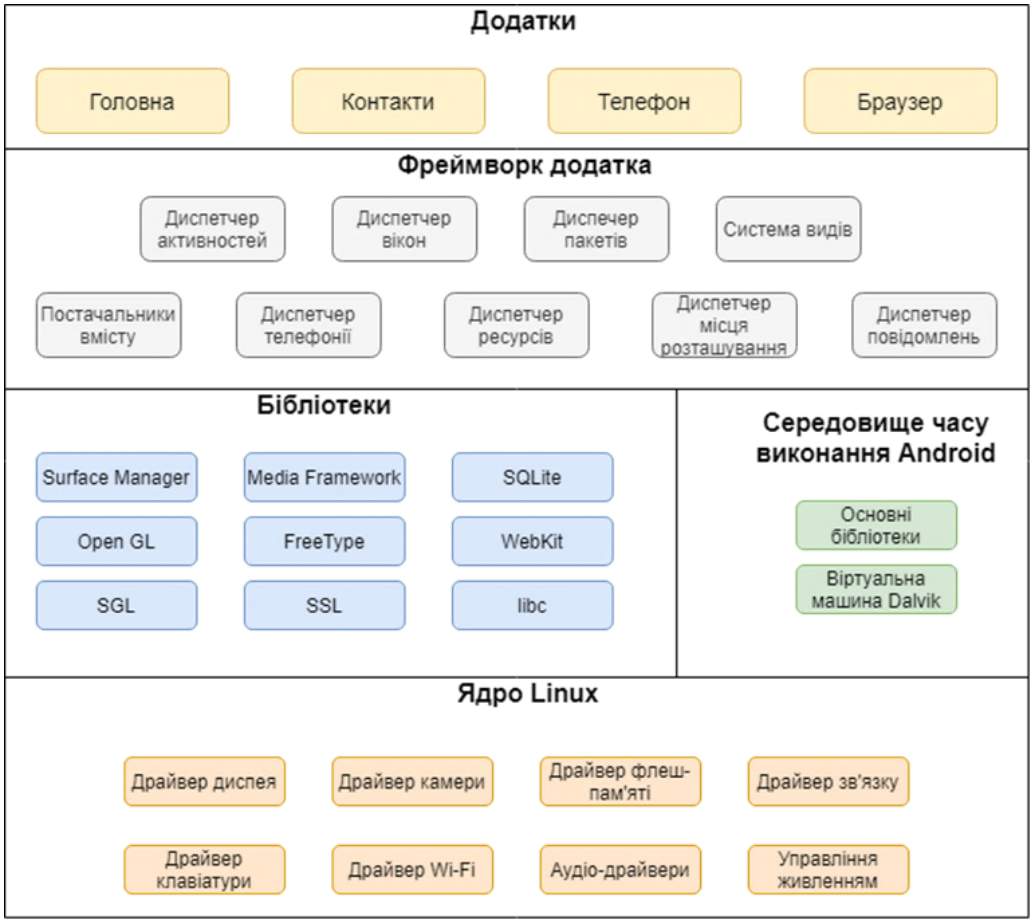

Рисунок 1.1 – Архітектура Android

З часу свого першого релізу у 2008 році Android пережила значні зміни та оновлення, кожне з яких має свою унікальну назву, пов'язану з якимось десертом (за винятком неактуальної Android 1.1).

I. Android 2.2 (Froyo). У травні 2010 року було випущено Android 2.2, відому також як «Froyo» або «заморожений йогурт». Ця версія принесла значні зміни, включаючи підтримку зовнішньої пам'яті, що дозволяла зберігати додатки на зовнішньому пристрої замість внутрішньої пам'яті Android-пристроїв. Крім цього, введення служби C2DM (Android Cloud to Device Messaging) надало розробникам можливість використовувати програми та дані, збережені в хмарі, тобто на віддалених серверах в Інтернеті, замість залежності від настільних комп'ютерів, ноутбуків або мобільних пристроїв. Ця технологія відкриває перед розробниками гнучкі можливості для налагодження обчислювальних ресурсів під конкретні потреби в будь-який момент часу, що робить їх більш ефективними та економічно вигідними [2].

Рисунок 1.2 – Android 2.2 Froyo

II. Android 2.3, відомий як «Gingerbread» або «імбирний пряник», був випущений у 2010 році та приніс багато нових можливостей для користувачів. Він включав покращену клавіатуру, поліпшену навігацію, ефективніше використання батареї та інші покращення. Крім цього, були введені нові інструменти для розробників, які дозволяли спростити створення додатків, включаючи можливість більш простого управління телефонними викликами та відповідями на них у додатках. Gingerbread також вніс істотні поліпшення в мультимедійні технології, включаючи нові API для роботи зі звуком та графікою, а також ігрові можливості, включаючи підвищену швидкість та нові датчики, такі як гіроскоп для обробки переміщень в просторі. Однією з найбільш значущих нововведень була підтримка технології NFC (Near-Field Communication) – бездротового високочастотного зв'язку для обміну даними між пристроями, які знаходяться на відстані декількох сантиметрів [2].

Рисунок 1.3 – Android 2.3 Gingerbread

III. Android 4.4 (KitKat). У жовтні 2013 року була представлена Android 4.4 KitKat, яка призначалася для оптимізації функціонування операційної системи на всіх типах Android-пристроїв, включаючи старі, обмежені пам'яттю. Ця версія принесла значні поліпшення та вирішила проблему «фрагментації» ринку Android. Фрагментація ринку – це ситуація, коли на ринку присутні різні версії операційної системи, що створює проблеми для розробників. Вони були змушені створювати додатки для різних версій ОС або обмежувати свою аудиторію. Android KitKat допомогла зменшити цю фрагментацію, що полегшило розробку та забезпечило доступність додатків для більшої кількості користувачів. Крім цього, KitKat внесла вдосконалення в графіку, мультимедійні можливості, засоби аналізу пам'яті та безпеку операційної системи.

Рисунок 1.4 – Android 4.4 KitKat

IV. Android 6.0 (Marshmallow). У вересні 2015 року була представлена Android Marshmallow, що вразила багатьма нововведеннями. Ця версія включила в себе ряд нових можливостей:

a) «Now on Tap» дозволила користувачам отримувати інформацію Google Now безпосередньо в контексті використаної програми.
b) «Doze» і «App Standby» спрямовані на збереження заряду батареї, що робить Android більш ефективним у керуванні енергією.
c) Нова модель дозволів спрощує установку додатків та керування доступом до приватних даних.
d) Додана підтримка аутентифікації за відбитком пальця для більшого рівня безпеки.
e) Покращено захист даних і підвищено рівень доступності.
f) Додано підтримку екранів 4K для вищої якості зображення.
g) Нові можливості для аудіо та відео, а також засоби роботи з камерою, включаючи режим «ліхтарик» та переробку зображень.

Android Marshmallow підняла планку функціональності та захисту, надаючи користувачам та розробникам інструменти для більш комфортного та безпечного використання мобільних пристроїв [2].

Рисунок 1.5 – Android 6.0 Marshmallow

V. Android 12 (Snow Cone). У вересні 2021 року світ побачив Android 12, відомий як «Snow Cone». Ця версія принесла чимало нововведень та поліпшень для користувачів та розробників. Однією з ключових особливостей Android 12 є поліпшена приватність та безпека. Впроваджено інноваційні засоби контролю доступу до особистих даних, включаючи мікрофон, камеру та місцезнаходження. Користувачі отримали більше контролю над даними, які додатки можуть збирати. Для поліпшення взаємодії та зручності використання, Android 12 вніс зміни в дизайн, включаючи новий інтерфейс «Material You». Він дозволяє користувачам налаштовувати кольорову палітру системи на основі свого власного стилю. Однією з функцій, яка призначена полегшити розробку додатків, є «Haptic Feedback API», яка дозволяє розробникам створювати відчуття відгуку під час взаємодії з додатками. Android 12 «Snow Cone» приніс низку інших нововведень, включаючи покращену підтримку відео- та аудіокодеків, розширену підтримку 5G мереж та інші покращення продуктивності та функціональності.

VI. Android 13 (Tiramisu). У 2022 році вийшов Android 13, відомий як «Tiramisu». Ця версія була спрямована на підвищення рівня інтеграції та зручності використання Android-пристроїв. Додатки стали більш інтегрованими з операційною системою, що дозволяє більш глибоку взаємодію між ними та спрощує спільне використання даних. Впроваджено покращену підтримку мультитач, що надає більше можливостей для розробників створювати інноваційні додатки. Нові інструменти для ефективного керування батареєю дозволяють продовжити роботу пристрою без зайвого споживання енергії. Android 13 «Tiramisu» також підняв безпеку на новий рівень, щоб забезпечити захист особистих даних користувачів та покращити стійкість до кіберзагроз.

VII. Android 14 (Upside Down Cake). Остання версія Android, Android 14, відома як (Upside Down Cake), була представлена у 2023 році. Ця версія відзначається багатьма інноваціями та функціональними покращеннями. Android 14 фокусується на поліпшенні взаємодії між користувачами та пристроями. Завдяки впровадженню передових голосових та жестових керувань, користувачі отримують більше можливостей для зручної та натуральної взаємодії зі своїми смартфонами та планшетами. Додатки стали ще більш інтегрованими, що спрощує обмін даними та функціями між ними. Відновлено фокус на безпеці та приватності користувачів, з розширеними інструментами для контролю за доступом до особистих даних. Зокрема, Android 14 включає в себе передові рішення щодо кіберзахисту, які зменшують ризики кібератак та допомагають зберегти конфіденційність даних. Ця версія продовжує доводити інновації та розвиток в світі мобільних технологій [3].

Android, операційна система, яка вже заполонила світ мобільних пристроїв, вражає своєю універсальністю. Вона виконується на різних типах пристроїв з різними розмірами та дозволами екрану. Не дивно, що використання програм на Android залежить від параметрів дисплея. Тому Google постачає Android з набором інструментів, що допомагають розробникам створювати додатки, адаптовані до різних характеристик екрану.

Політика компанії Google стосовно сумісності з пристроями ще більш жорстка. Вони приймають додатки, які можуть запускатися лише на сумісних пристроях. Наприклад, якщо додаток потребує фронтальну камеру, то в магазині Android Market він буде доступний тільки для тих пристроїв, які мають фронтальну камеру. Android автоматично розпізнає наявність незвичайного обладнання та відображає додаток лише на сумісних пристроях.

Щодо розробки додатків для Android, то варто відзначити, що вони пишуться на мові програмування Java. Проте це не повнофункціональний варіант Java, який використовується професійними розробниками при створенні додатків J2EE. Вміст Java для Android обмежений невеликою підмножиною, яка сумісна з віртуальною машиною Dalvik, спеціально розробленою для Android. Для оптимізації мобільних пристроїв з Java для Android були вилучені всі класи, які не мають сенсу на мобільних платформах [4].

У світі розвитку програмування Java виступає як ключовий актор, що несе історію та інновації. Ця мова та платформа історично визначили зміни в програмуванні та комп'ютерних технологіях.

Мова програмування Java була розроблена Джеймсом Гослінгом в 1990 році. Початково вона називалася «Oak», на честь дуба, який росте біля вікна офісу розробника. Однак згодом мові було дано ім'я Java, що стало символом її універсальності та привабливості.

З 1995 року Java стала вільно доступною, і ця подія відзначила нову еру в програмуванні. Розробники з усього світу швидко адаптувалися до цієї мови завдяки її модульному дизайну та можливості розширення її функціональності.

Основним структурним елементом мови Java є бібліотеки файлів, відомі як класи. Кожен клас містить невеликі фрагменти перевіреного коду, готового до виконання. Розробники можуть вбудовувати ці класи в нові програми, що робить розробку швидшою та більш ефективною. Модульна організація полегшує процес налагодження, адже виявлення помилок в невеликих модулях є набагато простішим.

Java вражає не лише як мова програмування, але і як платформа. Вихідний код Java пишеться в текстовому файлі з розширенням .java, і компілюється в файли .class за допомогою компілятора javac. Після цього програма виконується на віртуальній машині Java (Java VM), що дозволяє їй працювати на різних операційних системах без перекомпіляції [4].

Завдяки Java:

1) Розробники отримали зручну та потужну мову для створення програм.
2) Віртуальна машина Java забезпечує переносимість програм між платформами.
3) Широка бібліотека класів полегшує розробку та сприяє перевикористанню коду.
4) Модульний підхід спрощує розробку та підтримку програм.

Java залишається важливою мовою та платформою для розробників та їх проектів, і її вплив на світ програмування триває до сьогодні.

1.2 Огляд структури Android-прикладної програми

Платформа Android, здатна конкурувати з операційними системами настільних ПК за своєю функціональністю, пропонує багаторівневе середовище, побудоване на основі ядра Linux. Ця операційна система має величезний функціонал, який охоплює різні аспекти від вікон і уявлень до віджетів, спрямованих на відображення різноманітних елементів інтерфейсу, таких як редаговані поля, списки та розгортні списки.

У складі Android вбудований відкритий браузер із движком WebKit, який також використовується в браузері Safari на мобільних телефонах iPhone. Платформа підтримує різноманітні можливості підключення, включаючи Wi-Fi, Bluetooth і стільникові мережі (GPRS, EDGE, 3G тощо). Активно використовуються сервіси Google, такі як Google Maps для позиціонування пристрою [5].

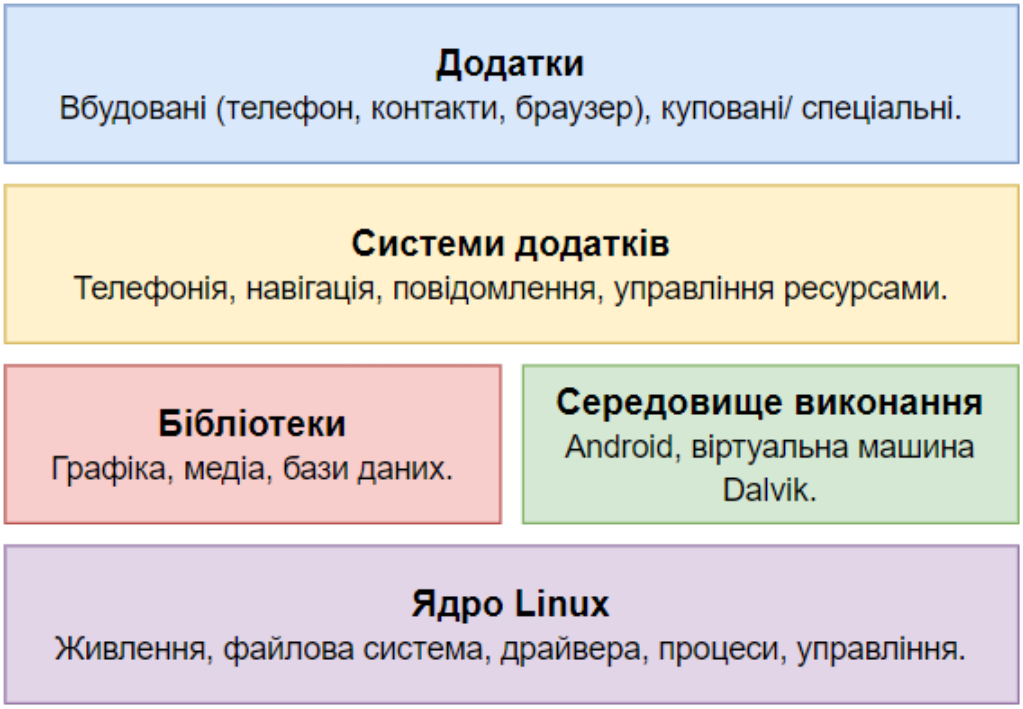

Рисунок 1.6 – Рівні програмного забезпечення Android

Однією з сильних сторін платформи є можливість обробки графіки та мультимедіа, а також зберігання даних. Зокрема, Android надає вбудовану підтримку 2D- та 3D-графіки через бібліотеку OpenGL, а також використовує відкритий SQLite для зберігання даних.

Враховуючи обмежені ресурси мобільних пристроїв, Android успішно вирішує проблеми, пов'язані з графікою, мультимедіа та зберіганням даних. Рисунок 1.6 демонструє спрощену схему рівнів програмного забезпечення Android, вказуючи на його комплексні можливості.

1.3 Огляд віртуальних машин Android

Операційна система Android заснована на ядрі Linux та використовує мову програмування Java для створення додатків. Прикладні програми виконуються у віртуальній машині, але варто відзначити, що вона не є віртуальною машиною Java (JVM). Замість цього, використовується відкрита технологія Dalvik Virtual Machine.

Під час запуску програми Android створює та запускає окремий екземпляр Dalvik VM, який розташований у межах керованого ядра Linux процесу. Dalvik є віртуальною машиною у складі операційної системи Android, використовується для виконання прикладних програм, зокрема на мобільних телефонах, планшетних комп'ютерах і смарт-телевізорах [6].

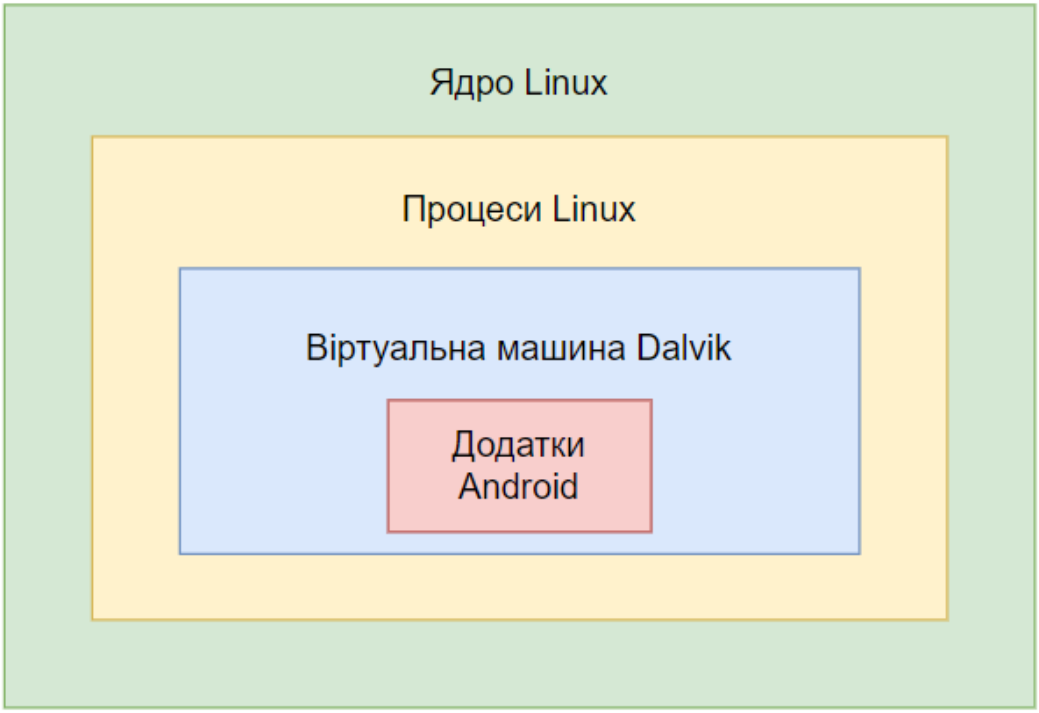

Рисунок 1.7 – Dalvik VM

Dalvik є частиною стека програмного забезпечення Android у версіях 4.4 KitKat та раніше. Він підтримує вбудовані та куповані/спеціальні додатки,

а також різні системні додатки, включаючи телефонію, навігацію, повідомлення та постачальників контенту.

Особливістю Dalvik є його відкритий вихідний код та використання у таких областях, як графіка, медіа, бази даних і віконний інтерфейс. Використовується також для обмежених систем з точки зору пам'яті та швидкості процесора.

Dalvik був розроблений Деном Борнстейном та названий на честь Dalvík в Ісландії. Програми для Android, головним чином написані на Java, компілюються в байт-код для віртуальної машини Java, який потім перетворюється в Dalvik байт-код і зберігається в .dex (Dalvik Executable) і .odex (оптимізоване Dalvik Executable) файлах.

Правонаступником Dalvik є Android Runtime (ART), який використовує той же байт-код і .dex файли, спрямований на підвищення продуктивності для кінцевих користувачів [7].

Android Runtime (ART) є середовищем виконання прикладних програм, використовуваним в операційній системі Android. ART забезпечує виконання байт-коду програм, перетворюючи його в інструкції, які виконуються за допомогою середовища виконання пристрою.

У версії Android 2.2 Froyo була впроваджена JIT-компіляція в Dalvik, що стало оптимізацією виконання програм за допомогою постійно профілюючих їх програм. Однак відмінності в роботі Dalvik в порівнянні з ART були помітною перевагою ART. ART використовує ahead-of-time (AOT) компіляцію, компілюючи всі прикладні програми у машинний код під час їхнього встановлення.

Однією з ключових відмінностей є те, що ART усуває інтерпретацію Dalvik, що базується на JIT-компіляції, що призводить до підвищення загальної ефективності виконання та зменшення енергоспоживання. Це сприяє підвищенню часу автономії батареї на мобільних пристроях. ART також пропонує швидше виконання програм, покращений розподіл пам'яті, ефективніші механізми збирання сміття (GC), нові засоби налагодження програм та точніше профілювання прикладних програм високого рівня.

З метою забезпечення зворотної сумісності, ART використовує той же вхідний байт-код, що і Dalvik, представлений у стандартних .dex файлах, які входять до складу APK файлів. Проте .odex файли замінюються Executable and Linkable Format (ELF) файлами. Після збирання та компіляції прикладної програми на пристрої, утиліта dex2oat запускається з виконуваного ELF-файла, спрощуючи виконання програм та усуваючи зайві накладні витрати, пов'язані з інтерпретацією Dalvik та трасуванням JIT-компіляції [7].

Технологія ART вимагає додаткового часу для компіляції програми під час її встановлення та забирає трошки більше простору для зберігання скомпільованого коду, переважно використовуючи флеш-пам'ять. З іншого боку, у версії Android 4.4 KitKat було впроваджено технологію попереднього огляду ART, яка стала альтернативою середовищу виконання та зберігання,

замість Dalvik віртуальної машини за вибором користувача. Пізніше, в Android 5.0 Lollipop, Dalvik було повністю замінено на ART.

Android-прикладна програма складається з різних типів елементів, таких як дії (Activities), що відображають графічний інтерфейс, програми з графічним інтерфейсом використовують дії, сервіси (Services) для тривалої роботи, джерела даних (Content providers) для управління доступом до даних, приймачі (Broadcast receivers) для реакції на події, такі як отримання повідомлень.

Програма розгортається на пристрої разом із файлом AndroidManifest.xml, який визначає необхідну інформацію про конфігурацію та інші деталі для встановлення програми. Цей файл також вказує дозволи, необхідні для роботи програми, забезпечуючи безпеку та запобігаючи можливому пошкодженню пристрою через некоректно написані програми [8].

1.4 Управління ресурсами мобільних пристроїв

Ядро, що використовується в Android, представляє собою модернізоване ядро серії Linux, призначене для задоволення особливих потреб мобільних платформ. Функції ядра в основному орієнтовані на драйвери, управління живленням, інструменти управління та корекцію обмежених можливостей Android. Особливо важливою є функціональність управління живленням для мобільних пристроїв.

Ядро Android є відкритим для загального доступу, і всі внесені зміни можна слідкувати через загальнодоступне сховище Android. У репозиторії доступні різні ядра, включаючи апаратно-специфічні для платформ, таких як MSM7xxx, Open Multimedia Application Platform (OMAP) та Tegra.

Зміни, які вносять у базову версію ядра, можна розподілити на кілька категорій: виправлення помилок, інструменти для підвищення простору для користувача (low memory killer, binder, ash mem, logger і т. д.), нова інфраструктура, особливо wake locks, підтримка нових SoCs (msm7k, msm8k і т. д.) та плат/пристроїв. В майбутньому специфічні ядра Android та базові ядра Linux мають об'єднатися, але цей процес йде повільно та займе час [8].

Android надає окремий адресний простір для кожної прикладної програми, спрямованої до користувача. Програми, які використовують Dalvik VM або ART, періодично переходять у сплячий режим або режим очікування. Це важливо, оскільки Android намагається якнайшвидше перевести систему в режим сну або очікування. Екран залишається включеним або процесор не переходить в сплячий режим, щоб швидше реагувати на пробудження. Інструменти Android для вирішення цього завдання використовують блокатори засипання, відомі як wakelocks. Функції wakelock можуть бути отримані компонентами ядра або адресним простором процесу користувача. Інтерфейс користувача для створення блокаторів

використовує файл /sys/power/wake_lock, в якому записується назва нового блокатора.

Для видалення блокування, процес записує назву файла в /sys/power/wake_unlock. Блокування можна також встановити на певний час, після якого воно автоматично відключиться. Всі системи, які використовують сервіс блокування на даний момент, вказані в файлі /proc/wakelocks. Інтерфейс ядра для блокаторів пробудження дозволяє вказати, чи має Wake_lock запобігати низькому енергоспоживанню або припиненню роботи системи.

Wake_lock створюється за допомогою процесу wake_lock_init() і вилучається з використанням wake_lock_destroy(). Створений блокатор може бути активований за допомогою wake_lock() та відключений за допомогою wake_unlock(). Також, як і в просторі користувача, можна визначити тайм-аут для блокування.

Концепція блокаторів глибоко інтегрована в Android у вигляді драйверів, і багато прикладних програм активно користуються ними.

1.5 API мобільних прикладних програм

Хоча більшість Android-програм написані мовою Java, існують значні відмінності між API Java та Android API. Основною різницею є те, що Android не використовує віртуальну машину Java; замість цього використовуються Dalvik або Android Runtime (ART).

Android-платформа не включає в себе віртуальну машину Java (Java VM), на якій виконується Java-байт-код. Замість цього класи Java компілюються у власний формат байт-коду, спеціально розроблений для віртуальної машини Dalvik. На відміну від віртуальних машин Java, які використовують стеки, Dalvik VM має архітектуру на основі регістрів.

Dalvik відрізняється від інших стандартних віртуальних машин за наступними особливостями:

1. VM було розроблено для ефективного використання менше пам'яті.
2. Змінено пул регістрів для використання лише 32-бітових індексів для спрощення інтерпретатора.
3. Використовує свій власний 16-бітний набір команд, що працює безпосередньо на локальних змінних величинах, натомість 8-розрядні команди стека.
4. Немає можливості завантажувати jar-файли, і існує інший порядок завантаження Android-бібліотек.
5. Обов'язкові властивості, визначені віртуальною машиною Java, можуть мати інше значення або бути непридатними в Android. Таким чином, хоча Android і використовує мову програмування Java, його архітектура та особливості відрізняються від традиційних програм, що використовують віртуальні машини Java [9].

1.6 Огляд архітектурних моделей для розроблення мобільних прикладних програм

Щороку зростає важливість розробки мобільних застосунків, але їхні життєві цикли відрізняються від тих, що характерні для настільних та веб-застосунків. Мобільні застосунки часто вимагають активної взаємодії з користувачем, навіть більше, ніж десктопні чи веб-застосунки. Вони, переважно, очікують від користувача визначених дій, таких як натискання кнопок, перед тим як відповісти оновленим інтерфейсом із потрібною інформацією.

У цьому підрозділі розглядається модель програмного забезпечення, зосереджена на інтерактивних аспектах мобільних застосунків. Також проводиться аналіз загальної концепції розробки мобільних застосунків.

Модель Model View Controller (MVC) визнана популярним підходом до розробки мобільних застосунків. Слід відзначити, що основна робота більшості мобільних застосунків полягає в отриманні даних зі сховища та оновленні інтерфейсу користувача із новою інформацією відповідно до запитів користувачів. Тому є сенс пов'язати компоненти інтерфейсу користувача з компонентами сховища даних. Однак, оскільки компоненти інтерфейсу користувача частіше оновлюються, щоб відповідати змінним вимогам користувачів та новим технологіям, ніж компоненти сховища даних, потрібно введення додаткових компонентів. Метою такої схеми є розподіл компонентів на компоненти інтерфейсу користувача (View), компоненти, які забезпечують основні функціональні можливості (Controller) та дані (Model).

Як було зазначено раніше, ключовими компонентами Android-прикладних програм є:

1. Активність – це основний клас інтерфейсу користувача.
2. Сервіс – пов'язаний із завданнями, що виконуються в фоновому режимі, такими як мережеві операції, не взаємодіючи при цьому з компонентами інтерфейсу користувача.
3. Постачальник контенту – надає можливість зберігання даних у програмі з використанням бази даних SQLite або SharedPreferences (дані, збережені у файлі XML на пристрої).
4. Широкомовний приймач – реагує на повідомлення системи, такі як попередження про низький рівень заряду, та забезпечує повідомлення користувача.

Зазвичай стандартний проект мобільної прикладної програми включає такий набір компонентів:

A. Компоненти інтерфейсу користувача.
B. Сервісні компоненти, які включають місце для відстеження місця розташування користувача за допомогою GPS, фонові завдання для отримання даних з інших служб та інші.
C. Компоненти даних, які використовуються для зберігання та отримання налаштувань [9].

D. Віджет-компоненти, такі як кнопки, поля для редагування тексту, поля для перегляду тексту, оброблення кліків користувачів і т.д.

Компоненти для інтерфейсу користувача в Android складаються з пакету Активностей, які, в сутності, подібні до компонент View в моделі MVC. Кожна Активність оновлює та модифікує інтерфейс для користувача протягом усього життєвого циклу програми. Компоненти постачальників та даних подібні до компонента Model, оскільки спостерігають за поведінкою даних. Вони також обробляють запити від View і Controller, щоб отримати інформацію про їхній стан та інструкції змінити його.

Компоненти віджетів аналогічні Controller, оскільки очікують введення від користувача, таке як натискання кнопки, і потім інтерпретують це введення та повідомляють Model або View для внесення змін.

Отже, можна зробити висновок, що на рівні стандартної мобільної програми модель MVC гармонійно вписується в систему Android, і кожен компонент програми в основному відображається в Model, View і Controller. Однак при ближчому розгляді деталей все виявляється не так просто. Наприклад, компонент Активностей, що є складовою програми, складається з Java-файлу та відповідного XML-файлу макета. Розробник визначає розташування UI-активності в файлі XML та реалізує функціональність та поведінку в файлі Java (розділ інтерфейсу користувача від логіки програми). При використанні віджет-компонентів спочатку слід визначити віджети як елементи XML-файлу макета для створення екземпляра віджета та його атрибутів. Оброблення поведінки віджета у відповідь на дії користувача також реалізується в файлі Java. Проте розробник має доступ до атрибутів віджета з макета XML під час реалізації його функціональності, і це порушує принцип розділу View від Model/Controller шаблону MVC [10].

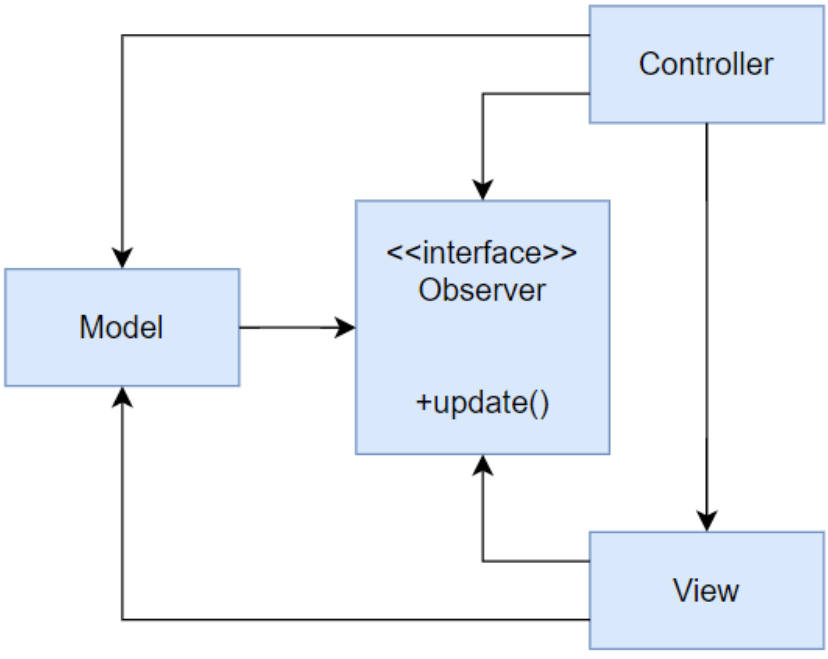

Рисунок 1.8 – Використання Observer у межах MVC

Ефективне рішення цієї проблеми може бути здійснене через використання шаблону Observer (Спостерігач) у концепції MVC. За цим підходом можна реалізувати інтерфейс Observer у компоненті Активності (View), щоб автоматично отримувати оновлення та повідомлення про будь-які зміни в логіці програми (стані Model).

При виникненні змін в стані програми, логіка додатку автоматично інформує всі Активності (спостерігачів) про зміни, відсунувши від них потребу в прямому взаємодії з моделлю. Цей підхід сприяє чіткому відокремленню між моделлю та представленням.

1.7 Огляд засобів розроблення мобільних прикладних програм

Набір розробки програмного забезпечення Android (SDK) включає в себе повний арсенал інструментів розробника, в який входять відладчик, бібліотеки, емулятор мобільного пристрою на основі QEMU, документація, приклади коду та навчальні матеріали [38]. На даний момент підтримуються платформи розробки, що працюють на операційній системі Linux (будь-який сучасний робочий стіл Linux), Mac OS X 10.5.8 або більш пізньої версії, а також Windows XP або більш пізньої версії.

Станом на березень 2015 року SDK не є доступним безпосередньо на платформі Android, але розробка програмного забезпечення можлива за допомогою спеціалізованих програм для Android (таблиця 1.1).

Таблиця 1.1. SDK Android

Розробник	Google
Перший випуск	Жовтень 2009 року
Стабільна версія	API levels 31, 32
Мова програмування	Java
Операційна система	Крос-платформна
Мова	Англійська
Тип	IDE, SDK
Сайт	developer.android.com/sdk/index.html

До кінця 2014 року офіційною інтегрованою середовище розробки (IDE) для Android був Eclipse (рис. 1.9). Для розроблення мобільних додатків використовувався спеціальний плагін – інструменти розроблення Android (ADT). З того часу середовище IntelliJ IDEA, що включає в себе всі випуски, надає повноцінну підтримку розробки Android «з коробки» (рис. 1.10). Крім того, NetBeans IDE також підтримує розробку Android через використання спеціального плагіна (рис. 1.11) [10].

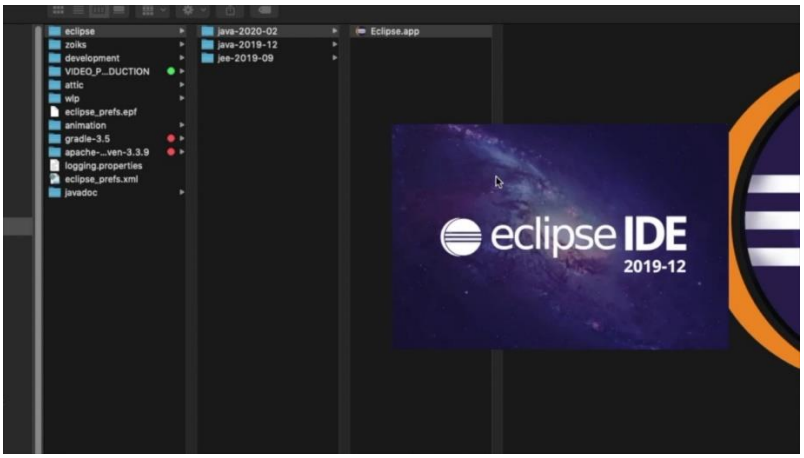

Рисунок 1.9 – IDE Eclipse

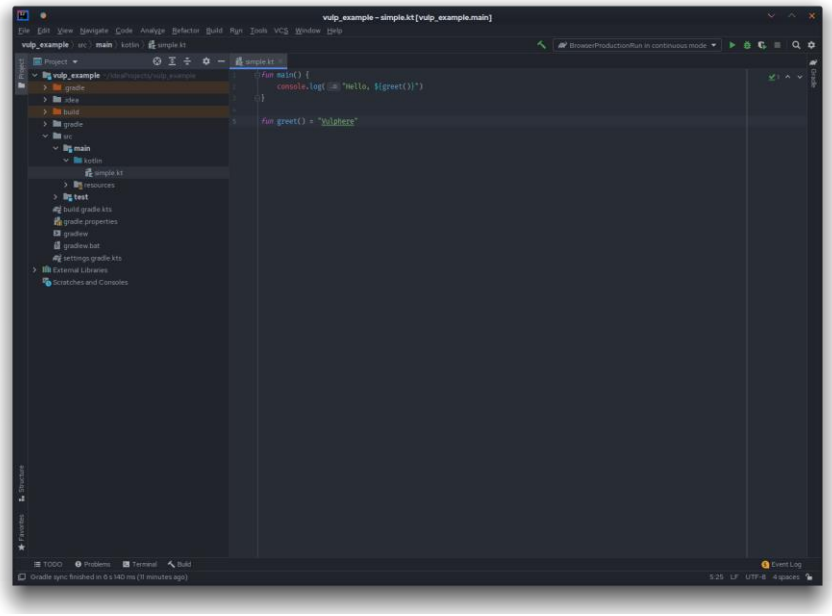

Рисунок 1.10 – IntelliJ IDEA

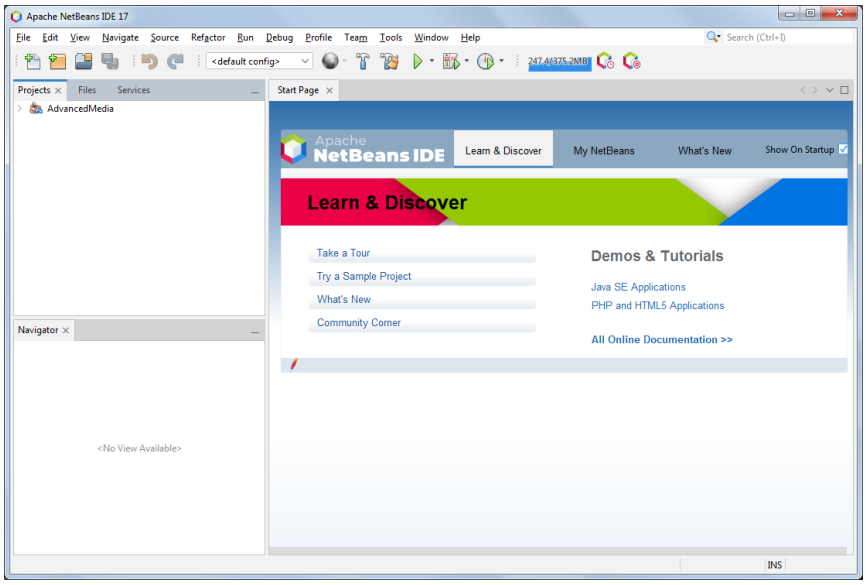

Рисунок 1.11 – NetBeans IDE

У 2015 році Google спільно з IntelliJ представили офіційне інтегроване середовище розробки від Google під назвою Android Studio. Однак розробники можуть використовувати інші інтегровані середовища розробки за своїм вибором. Також розробники можуть користуватися будь-яким текстовим редактором для редагування XML та Java-файлів, а потім використовувати інструменти командного рядка, такі як комплект розроблення Java і Apache Ant, для створення, компіляції та налагодження Android-додатків, а також для управління підключеними Android-пристроями, наприклад, викликанням перезавантаження, встановленням програмного забезпечення, видаленням пакетів.

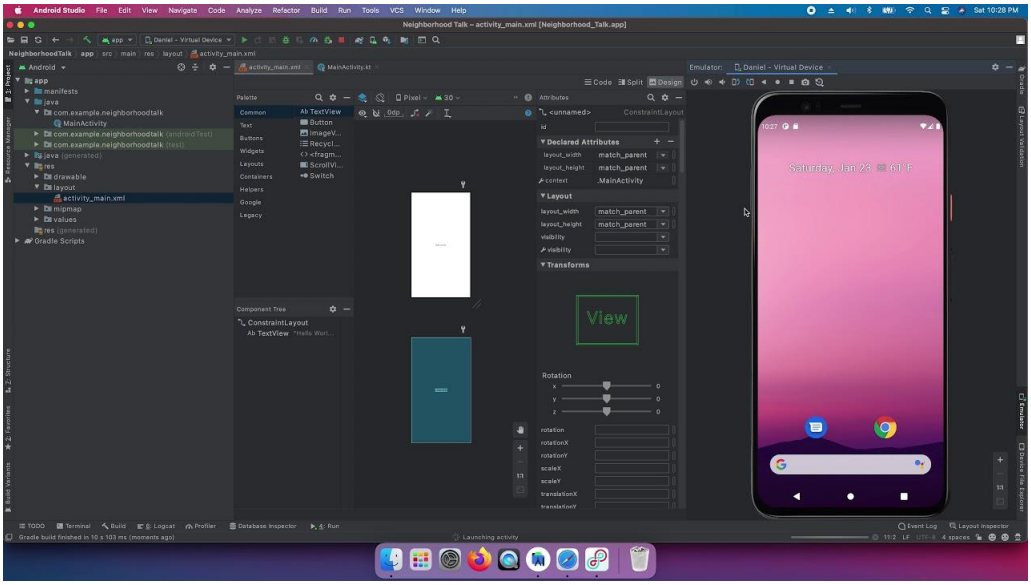

Рисунок 1.12 – Android Studio

Розвиток Android SDK іде паралельно з загальним ростом платформи Android. SDK також забезпечує підтримку старих версій Android, що дозволяє розробникам виконувати свої додатки на застарілих пристроях. Засоби розроблення є компонентами, які можна завантажувати, що дозволяє вам завантажити останні версії та платформи, а також старі версії та інструменти розроблення для тестування сумісності [10].

Додатки Android упаковуються у файли з розширенням .apk та зберігаються у папці /data/app на пристрої Android (ця папка доступна тільки для суперкористувача з погляду безпеки). Файли пакетів apk містять файли .dex (байт-код, який виконується виконавчою системою Dalvik VM), а також файли ресурсів та інше.

ADB (Android Debug Bridge) – це клієнт-серверна програма, яка забезпечує доступ до емулятора або пристрою. За допомогою ADB можна виконувати різні завдання, такі як копіювання файлів, встановлення скомпільованих програмних пакетів та виконання консольних команд. Через консоль можна змінювати налаштування журналу та взаємодіяти з базами даних SQLite, які знаходяться на пристрої. У попередніх версіях SDK

програма розташовувалася в папці tools, а тепер вона розміщена в папці platform-tools.

ADB складається з трьох компонентів: фонової служби (демона), яка працює в емуляторі; сервісу, який запускається на комп'ютері розробника; та клієнтської програми (наприклад, DDMS), яка взаємодіє зі службою через сервіс.

Android NDK (Android Native Development Kit) – це інструментарій для розробки на платформі Android, який дозволяє компілювати бібліотеки, написані на мовах програмування C, C++, та інших, в машинний код для архітектур ARM, MIPS або x86. Ці бібліотеки можуть бути встановлені на пристрої за допомогою Android Native Development Kit.

Рідні (native) класи, скомпільовані за допомогою NDK, можуть бути викликані з Java-коду, який виконується під управлінням Dalvik VM. Для цього використовується виклик System.loadLibrary, який є стандартним класом Java в середовищі Android (таблиця 1.2) [10].

Таблиця 1.2. Android Native Development Kit (NDK Android)

Розробник	Google
Перший випуск	Червень 2009 року
Стабільна версія	API levels 31, 32
Мова програмування	C та C++
Операційна система	Крос-платформна
Мова	Англійська
Тип	IDE, SDK
Сайт	developer.android.com/tools/sdk/ndk/index.html

Розроблені застосунки можуть бути скомпільовані та встановлені за допомогою традиційних інструментів розроблення. Проте відповідно до документації Android, використання Android Native Development Kit (NDK) не повинно обмежуватися лише розробкою додатків на мовах програмування C/C++. Розробники не повинні обирати NDK лише через власні вподобання до мови програмування, оскільки це може ускладнити структуру додатка, що врешті-решт може негативно вплинути на його продуктивність.

ADB-відладчик надає користувачеві root-права в Android Emulator, що дозволяє завантажувати та виконувати оптимізований під ARM, MIPS або x86 код. Компіляцію машинного коду можна виконати за допомогою GCC або Intel C++ Compiler на стандартному ПК. Проте запуск машинного коду на платформі Android ускладнений через використання нестандартної бібліотеки C (Libc), відомої як Bionic. Графічна бібліотека Android, використовувана для арбітражу та контролю за доступом до цього пристрою, називається Graphics Library Skia (SGL) і має відкриту ліцензію. Skia володіє двигунами як для Win32, так і для Unix-платформ, що дозволяє створювати крос-платформені додатки. Також, Skia є графічним двигуном, що лежить в основі веб-браузера Google Chrome [10].

Однак, відмінності від розробки Java-додатків, які базуються на таких інтегрованих середовищах розробки (IDE), як Eclipse, NDK базується на командному рядку і вимагає введення команд вручну для компіляції, розгортання та налагодження додатків. Інші інструменти сторонніх розробників можуть дозволяти інтегрувати NDK в Eclipse та Visual Studio.

1.8 Життєвий цикл візуальних компонентів

Activity – це ключовий компонент програми, який створює екран із інтерфейсом, що дозволяє користувачам взаємодіяти та виконувати різноманітні дії, такі як набір номера телефону, фотозйомка, відправка листа або перегляд карти. Кожній операції відведено вікно для відображення відповідного інтерфейсу користувача. Зазвичай вікно розгортається на весь екран, хоча його розмір може бути меншим, і може накладатися на інші вікна.

Програма, як правило, складається з кількох операцій, які можуть бути слабко пов'язані між собою. Зазвичай одну з операцій визначають як основну, яку пропонують користувачеві при першому запуску програми. У свою чергу, кожна операція може викликати іншу для виконання різних завдань. Кожен раз, коли запускається нова операція, попередня призупиняється, і система зберігає її в стек («стек переходів назад»). При запуску нової операції вона поміщається у стек переходів назад і відображається для користувача. Стек переходів назад працює за принципом «останній прийшов – перший вийшов», тому після завершення користувачем поточної операції та натискання кнопки «Назад», поточна операція видаляється зі стека (і знищується), і відновлюється попередня операція.

Коли операція призупиняється через запуск нової операції, для повідомлення про зміну її стану використовують методи зворотного виклику життєвого циклу операції. Існують кілька таких методів, які операція може приймати внаслідок зміни свого стану: створення операції, її призупинка, відновлення або знищення системою; крім того, кожен зворотний виклик дає можливість виконати певну дію, відповідну для певної зміни стану. Наприклад, у випадку призупинення операція повинна звільнити будь-які значні ресурси, такі як підключення до мережі або бази даних. Під час відновлення операції можна повторно отримати необхідні ресурси та відновити виконання перерваних дій. Ці зміни стану становлять частину життєвого циклу операції [11].

Для створення операції необхідно спочатку створити підклас класу Activity або скористатися наявним його підкласом. У цьому підкласі слід реалізувати методи зворотного виклику, які система викликає під час переходу операції між різними станами її життєвого циклу, такими як створення, зупинка, відновлення або знищення.

Два найбільш важливих методи зворотного виклику:

1. onCreate(): Цей метод обов'язково повинен бути реалізований, оскільки система викликає його під час створення операції. У його реалізації необхідно ініціалізувати ключові компоненти операції. Особливо важливим є виклик методу setContentView(), який визначає макет для інтерфейсу користувача операції.
2. onPause(): Цей метод викликається системою як перша ознака виходу користувача з операції (але це не означає, що операцію обов'язково буде знищено). Зазвичай в цьому методі застосовують будь-які зміни, які повинні бути збережені, за винятком поточного сеансу роботи користувача, оскільки користувач може не повернутися назад.

Крім цих двох методів, існують інші методи зворотного виклику життєвого циклу, які варто використовувати для забезпечення плавного переходу між операціями і оброблення непередбачених порушень у роботі операції, які можуть призвести до її зупинення або навіть знищення.

Для реалізації інтерфейсу користувача операції використовують ієрархію уявлень-об'єктів, отриманих із класу View. Кожне уявлення відповідає за певну прямокутну область вікна операції та може реагувати на дії користувачів. Наприклад, уявленням може бути кнопка, натискання на яку приводить до виконання певної дії.

У Android передбачено готовий набір уявлень, які можна використовувати для створення дизайну макета та його організації.

Віджети – це уявлення з візуальними (та інтерактивними) елементами, наприклад, кнопками, текстовими полями, чекбоксами або просто зображеннями.

Макети – це уявлення, отримані з класу ViewGroup, що забезпечують унікальну модель компонування для своїх дочірніх уявлень, таких як лінійний макет, сітка або відносний макет. Також можна створити підклас для класів View та ViewGroup (або скористатися наявними підкласами), щоб створити власні віджети та макети і потім застосувати їх до макета своєї операції [11].

Найчастіше для визначення макета використовують XML-файл макета, збережений у ресурсах додатка. Це дозволяє зберігати дизайн інтерфейсу користувача окремо від вихідного коду, який визначає поведінку операції. Метод setContentView() використовується для визначення макета інтерфейсу користувача операції, передаючи йому ідентифікатор ресурсу для макета. Однак також можна створити нові View у коді операції та створити ієрархію уявлень. Для цього потрібно вставити View у ViewGroup та використовувати цей макет, передаючи кореневий об'єкт ViewGroup у метод setContentView().

Для оголошення операції в маніфесті потрібно відкрити файл маніфесту та додти елемент <activity> як дочірній для елемента <application> (лістинг 1.1).

Лістинг 1.1 – Оголошення операції в маніфесті

```
<manifest ... >
    <application ... >
    <activity android:name=".ExampleActivity" />
    ...
</application ... >
    ...
</manifest >
```

Є кілька інших атрибутів, які можна додати до цього елемента для визначення різних властивостей операції, таких як мітка операції, значок операції або тема оформлення інтерфейсу користувача операції. Єдиним обов'язковим атрибутом є android:name, який вказує на назву класу операції. Після публікації додатка не рекомендується змінювати його назву, оскільки це може призвести до порушення деяких функціональних можливостей програми, наприклад, ярликів додатка.

Елемент <activity> також може визначати різні фільтри намірів за допомогою елемента <intent-filter>, щоб оголосити, як інші компоненти програми можуть активувати операцію [11].

Під час створення нової програми за допомогою інструментів Android SDK, у заготовці операції, яка створюється автоматично, міститься фільтр намірів, що оголошує операцію. Ця операція реагує на виконання основної дії, і її слід помістити в категорію переходу засобу запуску. Фільтр намірів має наступний вигляд (лістинг 1.2):

Лістинг 1.2 – Фільтр намірів

```
<activity android:name=".ExampleActivity"
    android:icon="@drawable/app_icon">
    <intent-filter>
        <action
android:name="android.intent.action.MAIN" />
        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Елемент <action> вказує, що це є основною точкою входу для додатку. Елемент <category> вказує на те, що цю операцію слід визначити як засіб якості додатків системи, щоб користувачі могли запускати цю операцію. Якщо має бути створено самодостатній додаток і заборонено іншим додаткам активувати його операції, тоді не потрібно створювати інших фільтрів намірів. У цьому випадку основна дія повинна бути представлена лише в одній операції, яку слід розмістити в категорію засобу запуску, як показано в прикладі вище.

Операції, які не мають бути доступні для інших додатків, не повинні включати фільтри намірів. Такі операції можна запустити самостійно за допомогою явних намірів.

Однак, якщо необхідно, щоб операція реагувала на неявні наміри, які надходять від інших додатків (а також з цього додатка), для операції необхідно визначити додаткові фільтри намірів. Для кожного типу наміру, на який має бути реакція, необхідно вказати об'єкт <intent-filter>, який містить елемент <action> і необов'язковий елемент <category> або <data> (або обидва ці елементи). Ці елементи визначають типи намірів, на які може реагувати операція [12].

Для ініціації запуску іншої операції достатньо викликати метод startActivity(), передаючи йому об'єкт Intent, який описує операцію, що потрібно запустити. У намірі можна вказати або конкретну операцію для запуску, або визначити тип операції, яку потрібно виконати (в такому випадку система вибере відповідну операцію, навіть якщо вона знаходиться в іншому додатку). Також намір може містити невеликий обсяг даних, які буде використовувати запущена операція.

Під час взаємодії з власним додатком, найчастіше потрібно лише запустити відповідну операцію. Для цього слід створити намір, що явно вказує на необхідну операцію за допомогою назви класу. Нижче наведено приклад запуску операції SignInActivity (лістинг 1.3):

Лістинг 1.3 – Одна операція запущена іншою операцією

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

Однак у додатку також може виникнути необхідність виконати певну дію, наприклад, надіслати лист, текстове повідомлення або оновити статус, використовуючи дані з операції. У такому випадку може бути відсутня відповідна функціональність в додатку, тому можна використовувати операції з інших додатків, які мають потрібні можливості та присутні на пристрої. Саме у цьому випадку наміри стають особливо корисними: можна створити намір, який описує необхідну дію, і система запускає його з іншої програми. При наявності кількох операцій, які можуть обробити цей намір, користувач може вибирати, яку з них використовувати. Наприклад, якщо користувач хоче надіслати електронний лист, можна створити відповідний намір (лістинг 1.4) [12]:

Лістинг 1.4 – Створення наміру з відправлення листа

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

Для завершення операції, достатньо викликати метод finish(). Також, щоб завершити окрему операцію, яка була запущена раніше, можна використовувати метод finishActivity().

У більшості випадків не рекомендується явно завершувати операцію за допомогою цих методів. Android система автоматично керує цим, і немає необхідності завершувати власні операції. Виклик цих методів може негативно вплинути на очікувану поведінку додатка. З ними слід користуватися тільки в тому випадку, коли абсолютно необхідно, щоб користувач не повертався до даного екземпляра операції.

Управління життєвим циклом операцій важливо під час розробки надійних і гнучких програм. Зв'язок операцій з іншими операціями, завданнями та стеком переходів назад має безпосередній вплив на їх життєвий цикл [13].

Існують три основні стани операцій:

1. Відновлена: Операція виконується на передньому плані екрана та відображається для користувача. Цей стан іноді називається «Виконується».
2. Припинена: Інша операція виконується на передньому фоні, але початкова операція залишається видимою, частково прозорою або неповністю покривається. Призупинена операція залишається активною, але система може завершити її в разі нестачі пам'яті.
3. Зупинена: Інша операція повністю перекриває початкову операцію, яка тепер виконується у фоновому режимі. Зупинена операція залишається активною, але користувач не бачить її, і система може завершити її при нестачі пам'яті.

Якщо операцію припинено або зупинено, система може очистити її з пам'яті, використовуючи метод finish(), або просто завершити процес операції. При повторному відкритті операції після її завершення, її потрібно повністю створити.

Весь життєвий цикл операції розгортається між викликами методів onCreate() та onDestroy(). У методі onCreate() операція виконує налаштування глобального стану, такі як визначення макета, та потім визволяє всі залишені ресурси у методі onDestroy(). Наприклад, якщо у операції працює фоновий потік для завантаження даних по мережі, операція може створити цей потік у методі onCreate() та зупинити його у методі onDestroy() [13].

Видимий життєвий цикл операції відбувається між викликами методів onStart() та onStop(). Протягом цього періоду операція відображається на екрані, де користувач може взаємодіяти з нею. Наприклад, метод onStop() викликається, коли запускається нова операція, а поточна більше не відображається. У проміжку між цими двома методами можна зберігати ресурси, необхідні для відображення операції для користувача. Наприклад, можна зареєструвати об'єкт BroadcastReceiver у методі onStart() для відстеження змін, що впливають на інтерфейс користувача, а потім скасувати його реєстрацію у методі onStop(), коли користувач більше не бачить

відображуваного. Протягом усього життєвого циклу операції система може кілька разів викликати методи onStart() та onStop(), оскільки операція може то відображатися для користувача, то приховуватися від нього.

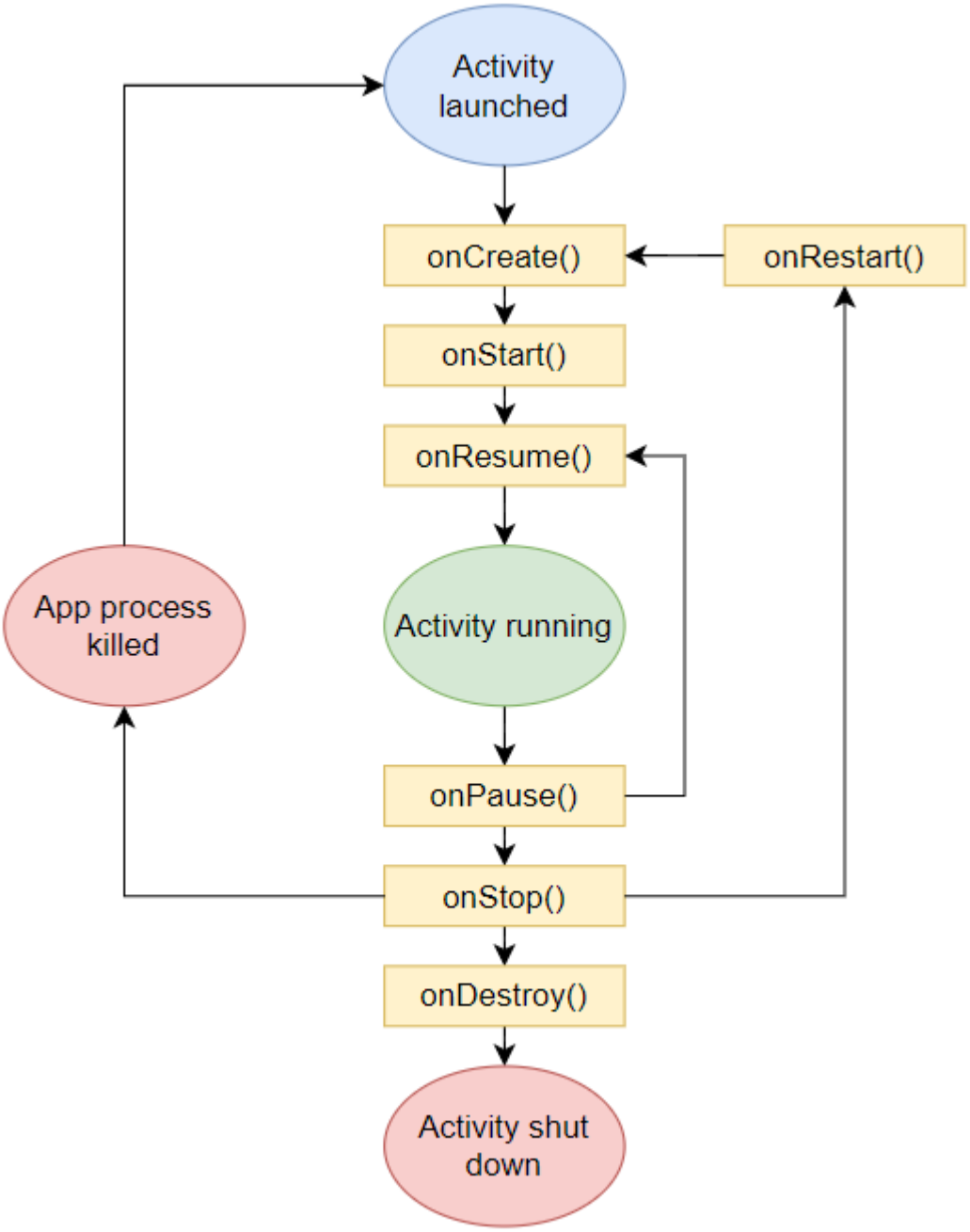

Рисунок 1.13 – Життєвий цикл операції

Життєвий цикл операції, яка активна на передньому плані, протікає між викликами методів onResume() та onPause(). Протягом цього періоду операція виконується на тлі всіх інших операцій і відображається для користувача. Операція може часто виходити в фоновий режим і повертатися з нього; наприклад, метод onPause() викликається під час переходу пристрою в сплячий режим або при з'явленні діалогового вікна. Оскільки перехід до цього стану може траплятися досить часто, код у цих двох методах повинен

бути легким, щоб уникнути повільних переходів та утримувати користувача в очікуванні.

На рисунку 1.13 проілюстровані проходи та шляхи, які операція може подолати між різними станами. Прямокутниками позначені методи зворотного виклику, які можна реалізувати для виконання дій між переходами операції від одного стану до іншого. Ці методи життєвого циклу перелічені в таблиці 1.3, де детально описано кожен метод зворотного виклику та вказано його місце в життєвому циклі операції загалом, включаючи інформацію про те, чи може система завершити операцію після виклику методу зворотного виклику [14].

Таблиця 1.3. Методи зворотного виклику життєвого циклу операції

Методи	Описи	Завершуючий	Наступний
onCreate()	Цей метод викликається при першому створенні операції. Тут потрібно налаштувати всі статичні елементи: створити представлення, прив'язати дані і т.д. Метод отримує об'єкт Bundle, який містить попередній стан операції (якщо такий стан було зафіксовано раніше). Завжди за цим методом слідує onStart().	Ні	onStart()
onRestart()	Цей метод викликається після призупинення операції, безпосередньо перед її повторним запуском. Після цього методу завжди слідує onStart().	Ні	onStart()
onStart()	Цей метод викликається безпосередньо перед тим, як операція стає видимою для користувача. Після цього методу може слідувати або onResume(), якщо операція переходить на передній план, або onStop(), якщо вона стає прихованою.	Ні	onResume() або onStop()
onResume()	Цей метод викликається безпосередньо перед тим, як операція починає взаємодію з користувачем. На цьому етапі операція перебуває вгорі стека операцій, і в неї надходять дані, введені користувачем. Після цього методу завжди слідує	Ні	onPause()

	onPause().		
onPause()	Цей метод викликається, коли система готується відновити іншу операцію. Зазвичай використовується для запису незбережених змін у постійне сховище даних, зупинки анімацій та інших елементів, які можуть використовувати ресурси ЦП та інші. Тут особливо важлива швидкодія, оскільки наступна операція не буде відновлена до тих пір, поки її не буде повернено на передній план. Після цього методу може слідувати або onResume(), якщо операція повертається на передній план, або onStop(), якщо операція стає прихованою для користувача.	Так	onResume() або onStop()
onStop()	Цей метод викликається, коли операція більше не відображається для користувача. Це може трапитися через те, що операцію знищено або внаслідок відновлення іншою операцією (наявною або новою). Після цього методу може слідувати або onRestart(), якщо операція відновлює взаємодію з користувачем, або onDestroy(), якщо операція переходить у фоновий режим.	Так	onRestart() onDestroy()
onDestroy()	Цей метод викликається перед тим, як операцію буде знищено. Це останній виклик, який отримує операція. Можна викликати його або через завершення операції (виклик методу finish()), або через тимчасове знищення системою цього екземпляра операції з метою вивільнення ресурсів. Щоб розрізнити ці два сценарії, використовується метод isFinishing().	Так	Нічого

У стовпці «Завершуючий» вказується, чи може система завершити процес, який включає операцію, після виконання методу, не враховуючи будь-якого іншого коду операції. Для трьох методів в цьому стовпці вказано «Так» (onPause(), onStop() та onDestroy()). Оскільки метод onPause() є першим із цих трьох після створення операції, він є останнім, який гарантовано викличеться перед тим, як система може завершити процес. Якщо системі терміново потрібно відновити пам'ять у випадку аварійної ситуації, то методи onStop() та onDestroy() не будуть викликані. Таким чином, рекомендується використовувати onPause(), щоб записати критично важливі дані (наприклад, редагування користувача) у сховище постійних даних. Важливо обережно вибирати інформацію, яку слід зберегти під час виклику onPause(), оскільки будь-яке блокування процедур у цьому методі може спричинити затримки у переході до наступної операції та призвести до неприємностей для користувача.

Методи, для яких у стовпці «Завершуючий» вказано «Ні», захищають процес, що містить операцію, від завершення відразу після їхнього виклику. Таким чином, завершити операцію можна між поверненням onPause() і викликом onResume(). Його не вдасться завершити, поки не буде знову викликано та повернено onPause() [14].

Перед тим як дозволити системі знищити операцію, вона викликає метод onSaveInstanceState(). У цей метод система передає об'єкт Bundle, який можна використовувати для збереження інформації про стан операції у вигляді пар «назва-значення». Для цього можна використовувати методи, такі як putString() та putInt(). У випадку завершення процесу вашого додатка і повернення користувача до операції, система повторно створює операцію та передає об'єкт Bundle в обидва методи: onCreate() та onRestoreInstanceState(). Обидва ці методи дозволяють витягти з об'єкта Bundle збережену інформацію про стан операції та відновити її. У випадку відсутності такої інформації об'єкт Bundle передається з нульовим значенням, що відбувається при створенні операції вперше.

У режимі виконання можливі зміни в деяких конфігураціях пристроїв, таких як орієнтація екрана, доступність клавіатури та мова. У таких ситуаціях Android повторно ініціює виконання рутинних завдань, спочатку викликаючи метод onDestroy(), а потім безпосередньо викликаючи метод onCreate(). Така поведінка дозволяє додатку адаптуватися до нових конфігурацій шляхом автоматичного перезавантаження альтернативних ресурсів, наприклад, різних макетів для різних орієнтацій та екранів різних розмірів.

Якщо операцію розроблено належним чином і вона належним чином підтримує перезапуск після зміни орієнтації екрана та відновлення свого стану, як описано раніше, додаток можна вважати більш стійким до інших непередбачених подій у життєвому циклі операції. Ефективний метод обробки такого перезапуску – збереження та відновлення стану операції за

допомогою методів onSaveInstanceState() та onRestoreInstanceState() (чи onCreate()).

Коли одна операція запускає іншу, обидві переходять через різні стани у своїх життєвих циклах. Перша операція припиняється та завершується (проте залишається видимою на фоні), тоді як друга операція стартує. У випадку, коли ці операції взаємодіють із збереженими на диску чи іншими даними, важливо розуміти, що перша операція не припиняється повністю до тих пір, поки друга не створиться. З іншого боку, процес запуску другої операції перекривається з зупинкою першої операції.

Чіткий порядок викликів у життєвому циклі визначається, особливо коли дві операції перебувають в одному процесі, і одна із них запускає іншу. Нижче подано порядок подій при запуску операції А операцією Б:

1. Метод onPause() операції А виконується.
2. Послідовно викликаються методи onCreate(), onStart() та onResume() операції Б (тепер користувач бачить операцію Б).
3. Якщо операція А більше не відображається на екрані, виконується метод onStop() для неї.

Ця передбачувана послідовність зворотних викликів у життєвому циклі дозволяє ефективно управляти переходом інформації від однієї операції до іншої. Наприклад, якщо після зупинки першої операції потрібно виконати запис в базу даних перед тим, як наступна операція зможе їх використати, то цей запис слід виконати під час виклику методу onPause(), а не в момент виклику методу onStop() [15].

1.9 Об’єкти Intent та фільтри об’єктів Intent

Intent використовується як об'єкт обміну повідомленнями для ініціювання виконання дій у компоненті іншої програми. Незважаючи на те, що об'єкти Intent полегшують обмін даними між компонентами у декількох аспектах, їх основне використання зосереджене на трьох ситуаціях:

1. Для запуску операції: Компонент Activity є екраном у додатку. Для ініціювання нового екземпляру компонента Activity, необхідно передати об'єкт Intent методу startActivity(). Об'єкт Intent визначає операцію, яку слід запустити, і містить всі необхідні дані. Якщо потрібно отримати результат після завершення операції, викликається метод startActivityForResult(), а результат передається через об'єкт Intent у метод onActivityResult() операції.
2. Для запуску служби: Компонент Service виконує дії у фоновому режимі без інтерфейсу користувача. Службу можна запустити для виконання одноразової дії (наприклад, завантаження файлу), передавши об'єкт Intent методу startService(). Об'єкт Intent описує службу, яку слід запустити, і містить всі необхідні дані. Якщо служба побудована з інтерфейсом клієнт-сервер, можна встановити прив'язування до неї з іншого компонента, передавши об'єкт Intent методу bindService().

3. Для розсилання широкомовних повідомлень: Широкомовне повідомлення – це повідомлення, яке може прийняти будь-який додаток. Система видаватиме різні широкомовні повідомлення про системні події, наприклад, коли система завантажується або пристрій починає заряджатися. Для відправлення широкомовного повідомлення іншим додаткам слід передати об'єкт Intent методам sendBroadcast(), sendOrderedBroadcast() або sendStickyBroadcast().

Існують два типи об'єктів Intent.

Явні об'єкти Intent вказують компонент, який потрібно запустити, за його назвою (повною назвою класу). Зазвичай їх використовують для запуску компонента з власного додатка, оскільки відомо назву класу операції чи служби, яку треба запустити. Наприклад, можна запустити нову операцію відповідно до дії користувача або запустити службу для завантаження файлу у фоновому режимі [15].

Неявні об'єкти Intent не містять назву конкретного компонента. Замість цього вони взагалі оголошують дію, яку треба виконати, що дає можливість компоненту з іншої програми обробити цей запит. Наприклад, якщо потрібно показати користувачеві місце на карті, то за допомогою неявного об'єкта Intent можна запросити інший додаток, який має відповідну функцію.

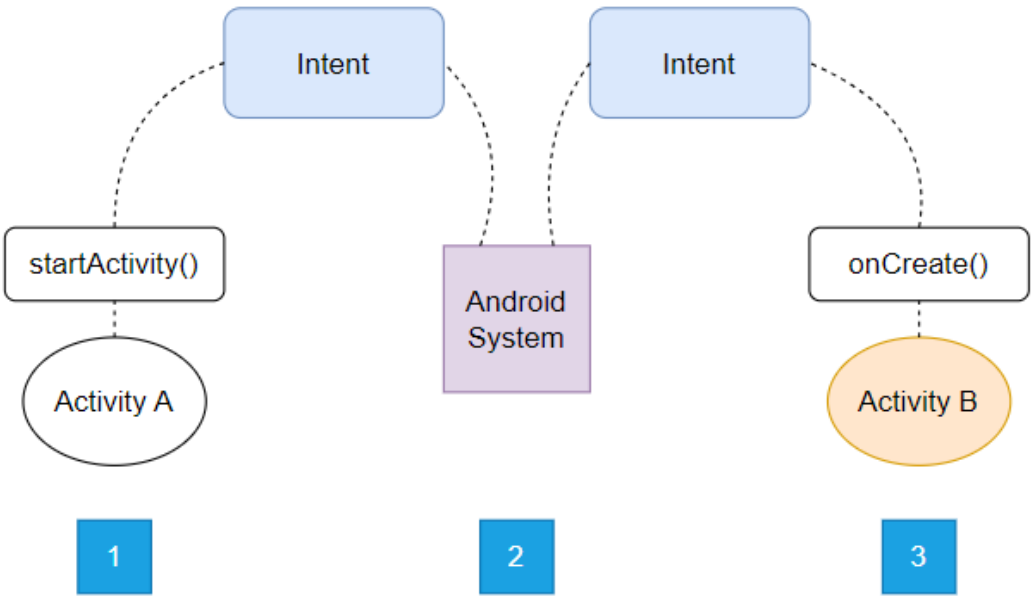

Рисунок 1.14 – Процес передачі неявного об’єкта Intent по системі для запуску іншої операції

При створенні явного об'єкта Intent для запуску операції або служби в системі Android відбувається наступне:

1. Операція А створює об'єкт Intent із описом дії та передає його методу startActivity().
2. Система Android шукає у всіх додатках фільтри Intent, які відповідають цьому об'єкту Intent.

3. Коли знайдено додаток із відповідним фільтром, система запускає відповідну операцію (Операцію B), викликаючи її метод onCreate() і передаючи йому об'єкт Intent (рисунок 1.14) [16].

Коли створено неявний об'єкт Intent, система Android визначає відповідний компонент, порівнюючи вміст об'єкта Intent із фільтрами Intent, які визначені в маніфест-файлах інших додатків на пристрої. Якщо об'єкт Intent відповідає фільтру Intent, система запускає цей компонент і передає йому об'єкт Intent. У випадку, якщо існують кілька фільтрів Intent, система відображає діалогове вікно, де користувач може вибрати програму для обробки запиту.

Фільтр Intent є виразом у файлі маніфесту програми, який визначає типи об'єктів Intent, які може приймати компонент. Наприклад, оголошення фільтру Intent для операції дозволяє іншим програмам безпосередньо запускати цю операцію за допомогою відповідного об'єкта Intent. Зворотно, якщо фільтри Intent для операції не оголошено, то запустити її можна лише за допомогою явного об'єкта Intent.

Об'єкт Intent включає інформацію, на основі якої система Android визначає, який компонент слід запустити. Ця інформація включає точну назву компонента або категорію компонентів, які повинні обробити цей об'єкт Intent. Також в об'єкті Intent містяться дані, які необхідні компоненту-одержувачу для належного виконання дії, такі як виконувана дія та дані, з якими цей компонент має працювати.

Основні відомості, які можуть бути включені в об'єкт Intent:

1. Назва компонента.

Це назва компонента, який має бути запущений. Ця інформація є необов'язковою, але саме вона робить об'єкт Intent явним. Якщо ця назва присутня, об'єкт Intent передається тільки компоненту програми, який визначений за цією назвою. У випадку відсутності назви компонента, об'єкт Intent стає неявним, і система визначає, який компонент отримає цей об'єкт Intent на основі інших відомостей, що містяться в ньому. Таким чином, якщо метою є запуск конкретного компонента з власного програмного додатка, слід вказати його назву.

2. Дія.

Дія – це рядок, який визначає стандартну операцію, яку слід виконати (наприклад, «перегляд» або «вибір»). У контексті об'єктів Intent, видачі широкомовних повідомлень, дія вказує на конкретну подію, яка сталася і про яку повідомляється. Дія в значній мірі визначає структуру іншої інформації в об'єкті Intent, зокрема, вміст розділу даних і додаткових даних.

Для використання об'єктів Intent в межах власного додатка або для виклику компонентів з інших додатків слід вказати власні дії. Проте, зазвичай рекомендується використовувати константи дій, визначені класом Intent або іншими класами платформи. Кілька стандартних дій для запуску операції [16]:

ACTION_VIEW: використовується в об'єкті Intent із методом startActivity(), коли є конкретна інформація, яку можна відобразити користувачеві, наприклад, фотографія в додатку галереї або адреса для перегляду в картографічному додатку.

ACTION_SEND: цю дію часто називають «share» (намір надати загальний доступ). Використовується в об'єкті Intent із методом startActivity(), коли є певні дані, доступ до яких користувач може надати через інший додаток, наприклад, додаток для роботи з електронною поштою або соціальними мережами.

Інші константи, які визначають стандартні дії, можна знайти у довіднику за класом Intent. Також, інші дії визначаються в інших частинах платформи Android, наприклад, у «Settings» визначаються дії, що відкривають екрани програми для налаштувань системи.

Дію можна вказати для об'єкта Intent за допомогою методу setAction() або конструктора Intent. При визначенні власних дій слід обов'язково використовувати як префікс їхнього імені назву пакета додатка. Наприклад (лістинг 1.5):

Лістинг 1.5 – Встановлення назви пакета під час визначення дії

```
static final String ACTION_TIMETRAVEL =
"com.example.action.TIMETRAVEL";
```

3. Дані.

Дані в об'єкті Intent представлені URI (об'єкт Uri), який вказує на дані, що підлягають обробці, а також тип MIME цих даних. Зазвичай тип даних визначається конкретною дією об'єкта Intent. Наприклад, якщо дією є ACTION_EDIT, то дані повинні містити URI документа, який планується редагувати.

При створенні об'єкта Intent, крім URI, часто необхідно вказати тип даних (їх MIME-тип). Наприклад, операція, яка призначена для відображення зображення на екрані, ймовірно, не зможе коректно відтворити аудіофайл, навіть якщо обидва мають однаковий формат URI. Тому вказання MIME-типу даних допомагає системі Android визначити найбільш підходящий компонент для обробки об'єкта Intent. Зазначено, що тип MIME іноді може бути успадкований від URI, особливо, якщо дані представлені як content: URI, що вказує, що дані розташовані на пристрої та керуються ContentProvider. Це дозволяє системі автоматично визначити тип MIME даних.

Для встановлення тільки URI-даних використовується метод setData(). Для встановлення тільки типу MIME використовується метод setType(). При необхідності обидва параметри можна встановити явно за допомогою методу setDataAndType() [17].

4. Категорія.

Категорія в об'єкті Intent представляє собою рядок, що містить додаткові відомості про те, яким компонентом має бути оброблено виклик цього об'єкта Intent. В об'єкт Intent можна включити будь-яку кількість описів категорій, але для більшості об'єктів Intent категорії не є обов'язковими. Нижче наведено кілька стандартних категорій:

CATEGORY_BROWSABLE: ця категорія дозволяє цільовій операції запускати себе у веб-браузері для відображення даних, зазначених за посиланням, таких як зображення або повідомлення електронної пошти.

CATEGORY_LAUNCHER: ця категорія вказує, що операція є початковою для завдання; вона призначена для використання в інструментах запуску системи для визначення початкового додатка.

5. Додаткові дані.

Додаткові дані в об'єкті Intent представлені парами «ключ-значення», які містять іншу інформацію, необхідну для виконання конкретної дії. Так само, як деякі дії використовують певні види URI-даних, інші дії можуть вимагати конкретних додаткових даних.

Додавання додаткових даних до об'єкта Intent можна виконати різними методами, такими як putExtra(), який приймає два параметри: назва та значення ключа. Також можна створити об'єкт Bundle, який містить всі додаткові дані, і потім встановити цей об'єкт Bundle в об'єкт Intent за допомогою методу putExtras().

Наприклад, при створенні об'єкта Intent для надсилання листів за допомогою методу ACTION_SEND, можна вказати отримувача за допомогою ключа EXTRA_EMAIL, а тему повідомлення – за допомогою ключа EXTRA_SUBJECT. Клас Intent має багато констант EXTRA_* для стандартних типів даних. Якщо потрібно оголосити власні додаткові ключі (для об'єктів Intent, які обробляє додаток), слід вказати як префікс назву пакета цього додатка. Наприклад (лістинг1.6) [17]:

Лістинг 1.6 – Встановлення назви пакета додатка

```
static final String MY_PHOTOMATH =
"com.example.MY_PHOTOMATH";
```

Використання фільтра Intent являє собою неефективний метод уникнення запуску компонентів іншими додатками. Навіть після застосування фільтрів Intent компонент може реагувати лише на неявні об'єкти Intent певного виду, і теоретично інший додаток може ініціювати запуск компонента за допомогою явного об'єкта Intent, якщо розробник визначить назви компонентів. Якщо важливо, щоб лише власний додаток міг запускати один із компонентів, слід встановити для атрибута exported цього компонента значення false.

Неявний об'єкт Intent перевіряється фільтром через порівняння з кожним із трьох його елементів. Щоб об'єкт Intent був переданий

компоненту, він повинен успішно пройти всі три тести. Якщо він не відповідає хоча б одному з них, система Android не передасть цей об'єкт Intent компоненту. Тим не менш, оскільки компонент може мати кілька фільтрів Intent, об'єкт Intent, який не пройшов через один з фільтрів компонента, може пройти через інший фільтр.

1.10 Служби та сервіси мобільних платформ

Service є складовою програми, призначеною для виконання тривалих операцій у фоновому режимі і не обладнаною інтерфейсом користувача. Інші компоненти програми можуть запускати сервіс, який продовжує свою роботу в фоновому режимі, навіть якщо користувач перейде до іншого додатку. Крім того, компонент може прив'язатися до сервісу для взаємодії з ним і навіть виконувати міжпроцесову взаємодію (IPC). Наприклад, сервіс може обробляти мережеві транзакції, відтворювати музику, виконувати введення-виведення файлу або взаємодіяти з постачальником контенту, все це у фоновому режимі [17].

Фактично сервіс може мати дві форми:

1. Запущена: Сервіс вважається «запущеним», коли компонент додатка (наприклад, операція) запускає його за допомогою методу startService(). Після запуску сервіс може функціонувати у фоновому режимі протягом необмеженого часу, навіть після знищення компонента, що його ініціював. Зазвичай запущений сервіс виконує лише одну операцію і не повертає результати викликаючому компоненту, наприклад, завантаження файлу по мережі. Коли операцію завершено, сервіс автоматично зупиняється.
2. Прив'язана: Сервіс вважається «прив'язаним», коли компонент програми з'єднується з ним за допомогою методу bindService(). Прив'язаний сервіс надає інтерфейс клієнт-сервер, який дозволяє компонентам взаємодіяти з сервісом, надсилати запити, отримувати результати, а також виконувати це між різними процесами через міжпроцесову взаємодію (IPC). Прив'язаний сервіс працює лише поки до нього прив'язаний інший компонент програми. До сервісу може бути прив'язано кілька функцій одночасно, проте коли всі вони відмовляються від прив'язування, сервіс автоматично знищується.

Для створення служби слід створити підклас класу Service або скористатися одним із наявних його підкласів. У реалізації слід визначити деякі методи зворотного виклику, які будуть обробляти ключові етапи життєвого циклу служби та, за необхідності, забезпечувати механізм прив'язування компонентів.

Найважливіші методи зворотного виклику, які необхідно реалізувати, включають:

onStartCommand(): цей метод викликається системою, коли інший компонент, наприклад, операція, видає запит на запуск цієї служби через startService(). Після виконання цього методу служба стає активною і може працювати у фоновому режимі протягом необмеженого часу. Якщо вже реалізовано цей метод, службу можна зупинити за допомогою stopSelf() або stopService(). Однак, якщо просто потрібно забезпечити прив'язування, реалізація цього методу є необов'язковою [17].

onBind(): система викликає цей метод, коли інший компонент виражає бажання прив'язатися до служби, наприклад, для виконання віддаленого виклику за допомогою bindService(). У реалізації цього методу слід надати інтерфейс, який клієнти використовують для взаємодії зі службою, повертаючи IBinder. Цей метод завжди повинен бути реалізований, проте, якщо прив'язування не дозволено, слід повертати значення null.

onCreate(): цей метод викликається системою під час першого створення служби для виконання одноразових процедур налаштування перед викликом onStartCommand() або onBind(). Якщо служба вже запущена, цей метод не викликається.

onDestroy(): цей метод викликається системою, коли служба більше не використовується і буде знищена. Служба повинна реалізувати цей метод для очищення ресурсів, таких як потоки, зареєстровані приймачі, ресивери та інші. Це останній виклик, який отримує служба.

Система Android автоматично припиняє роботу служби лише тоді, коли виявляється недостатньо пам'яті і необхідно відновити системні операції, що активні на передньому плані. У випадку, коли служба пов'язана з операцією на передньому плані, її видалення стає менш ймовірним, і якщо службу призначено для роботи в фоновому режимі (як раніше обговорено), ймовірність її видалення практично мінімальна.

У інших випадках, якщо служба була запущена і триває, система з часом зменшує її пріоритет в списку фонових завдань, і служба стає вразливою до можливості її видалення. У разі активної роботи служби слід передбачати докладну обробку її перезапуску системою.

Коли система вирішує припинити роботу служби, вона запускає її знову, як тільки знову доступно достатньо ресурсів [17].

Служба, що була запущена, виникає тоді, коли інший компонент викликає метод startService(), викликаючи метод onStartCommand() служби.

Під час роботи служба має свій термін життя, незалежний від компонента, який її запустив, і може функціонувати у фоновому режимі протягом необмеженого періоду, навіть якщо компонент, що її запустив, був знищений. Тому після виконання своєї роботи служба повинна зупинятися автоматично за допомогою методу stopSelf() або може бути зупинена іншим компонентом за допомогою методу stopService().

Компонент програми, такий як операція, може активувати службу, викликаючи метод startService() і передаючи об'єкт Intent, який вказує службу

та будь-які дані, які служба повинна використовувати. Служба отримує цей об'єкт Intent у методі onStartCommand().

Якщо, наприклад, операції потрібно зберегти дані в мережевій базі даних, вони можуть запустити службу, передаючи дані через метод startService(). Служба отримує цей намір у методі onStartCommand(), встановлює з'єднання з Інтернетом та виконує транзакцію з базою даних. Після завершення транзакції служба автоматично зупиняється та видаляється.

У звичайній практиці існують два базові класи, які можна використовувати для створення запущеної служби:

1. Service: Це основний клас для всіх служб. При успадкуванні від цього класу важливо створити новий потік, в якому буде виконуватися весь функціонал служби. За замовчуванням служба використовує основний потік додатка, що може уповільнити будь-яку операцію, яку виконує додаток.
2. IntentService: Це підклас класу Service, який використовує робочий потік для обробки всіх запитів запуску послідовно. Цей варіант є оптимальним, якщо не потрібно, щоб служба обробляла декілька запитів одночасно. Достатньо реалізувати метод onHandleIntent(), який отримує намір для кожного запиту запуску, дозволяючи виконувати фонову роботу.

Оскільки більшості запущених додатків не потрібно обробляти декілька запитів одночасно, що може бути потенційно небезпечним, рекомендується використовувати клас IntentService для реалізації служби. Цей підклас класу Service пропонує кілька переваг:

1. Робочий потік за замовчуванням.

IntentService автоматично створює робочий потік, який виконує всі наміри, передані в метод onStartCommand(). Це дозволяє виконувати операції паралельно, не впливаючи на основний потік додатка.

2. Робоча черга для намірів.

Створюється робоча черга, яка передає наміри по одному в метод onHandleIntent(). Це гарантує безпеку багатопотоковості і запобігає конфліктам при обробці запитів одночасно.

3. Автоматична зупинка служби.

IntentService автоматично зупиняється після оброблення всіх запитів запуску, уникавши необхідності вручну викликати stopSelf().

4. Невизначений метод onBind().

Метод onBind() повертає null, оскільки IntentService не призначений для прив'язування.

5. Реалізація onStartCommand() за замовчуванням.

IntentService надає реалізацію методу onStartCommand(), яка автоматично відправляє намір у робочу чергу та подальшу обробку в метод onHandleIntent() [17].

Все це означає, що для реалізації служби досить реалізувати лише метод onHandleIntent(), який виконує роботу, передану клієнтом. Якщо вирішено визначити додаткові методи зворотного виклику, такі як onCreate(), onStartCommand() або onDestroy(), слід обов'язково викликати реалізацію суперкласу, щоб забезпечити правильне оброблення життєвого циклу робочого потоку. Детальний приклад реалізації класу IntentService подано в лістингу 1.7.

Лістинг 1.7 – Реалізація класу IntentService

```
public class HelloIntentService extends IntentService {
            /**
             * Потрібен конструктор, він має викликати
IntentService(String)
             * конструктор з назвою для робочого
потоку.
             */
            public HelloIntentService() {
                super("HelloIntentService");
            }
             /**
             * IntentService викликає цей метод із
робочого потоку за замовчуванням.
             * Коли цей метод виконався, IntentService
             * зупиняє службу, за необхідності.
             */
            @Override
            protected void onHandleIntent(Intent
intent) {

                long endTime =
System.currentTimeMillis() + 5*1000;
                while (System.currentTimeMillis() <
endTime) {
                    synchronized (this) {
                        try {
                            wait(endTime -
System.currentTimeMillis());
                        } catch (Exception e) {
                        }
                    }
                }
            }
}
```

Запущена служба повинна самостійно керувати своїм життєвим циклом. Іншими словами, система не автоматично зупиняє або не знищує службу, якщо необхідно відновити системну пам'ять. Після повернення з методу onStartCommand() служба продовжує свою роботу. Таким чином, для припинення роботи служби вона повинна викликати метод stopSelf(), або інший компонент може зупинити її, викликавши метод stopService() [18].

Коли служба отримує запит на зупинку через stopSelf() або stopService(), система якнайшвидше знищує службу. Проте, якщо служба обробляє кілька запитів onStartCommand() одночасно, не рекомендується зупиняти службу після завершення оброблення першого запиту. Це може призвести до проблем, оскільки може з'явитися новий запит на запуск, і зупинка служби в кінці першого запиту може вплинути на наступний. Щоб уникнути цієї ситуації, можна використовувати метод stopSelf(int).

Метод stopSelf(int) гарантує, що запит на зупинку служби завжди ґрунтується на останньому запиті запуску. При виклику stopSelf(int) передається ідентифікатор запиту запуску (startId), який отримується в onStartCommand(). Таким чином, якщо служба отримає новий запит запуску до виклику stopSelf(int), вона не буде зупинена, і запит на зупинку буде спрямований лише до відповідного запиту запуску.

Прив'язана служба є службою, яка дозволяє компонентам програми прив'язуватися до неї шляхом виклику методу bindService(). Це сприяє створенню тривалого з'єднання та, як правило, обмежує можливість її запуску за допомогою методу startService().

Створення прив'язаної служби виправдано, коли потрібно взаємодіяти з операціями та іншими компонентами додатка або надавати функціонал іншим додаткам за допомогою міжпроцесної взаємодії (IPC) [18].

Для створення прив'язаної служби необхідно реалізувати метод зворотного виклику onBind(), який повертає об'єкт IBinder, що визначає інтерфейс взаємодії зі службою. Після цього інші компоненти додатка можуть викликати метод bindService(), щоб отримати інтерфейс та взаємодіяти зі службою.

Прив'язана служба існує тільки для обслуговування компонента, який до неї прив'язаний. Коли немає прив'язаних компонентів, система знищує службу автоматично, не вимагаючи вручну зупиняти прив'язану службу, як це потрібно для служби, запущеної за допомогою методу onStartCommand().

Щоб створити прив'язану службу, перш за все, необхідно визначити інтерфейс взаємодії між службою та клієнтом. Цей інтерфейс має бути реалізацією об'єкта IBinder, який служба повертає з методу зворотного виклику onBind(). Коли клієнт отримує об'єкт IBinder, він може почати взаємодію зі службою через цей інтерфейс.

Декілька клієнтів може бути прив'язано до служби одночасно. Коли клієнт завершує взаємодію зі службою, він викликає метод unbindService() для скасування прив'язування. Як тільки жоден клієнт не залишається прив'язаним до служби, система автоматично знищує її.

Після запуску служба може інформувати користувача про події за допомогою двох основних методів: Випливних повідомлень та Повідомлень у рядку стану.

Випливне повідомлення – це короткочасне сповіщення, яке з'являється на поверхні поточного вікна. З іншого боку, Повідомлення у рядку стану представляє собою значок, який з'являється у рядку стану і містить повідомлення, яке користувач може обрати для виконання певної дії, наприклад, запуску певної операції.

Зазвичай використання Повідомлень у рядку стану є найзручнішим рішенням у випадках завершення фонових операцій, наприклад, завантаження файлу. Після завершення таких процесів користувач може прийняти необхідні дії, обравши відповідне повідомлення у рядку стану. Коли користувач розглядає розширений вигляд повідомлення, він може ініціювати операцію, наприклад, перегляд завантаженого файлу.

Служба переднього плану – це активна служба, з якою користувач взаємодіє, і, отже, вона виключена з розгляду для видалення системою в разі обмеження пам'яті. Служба переднього плану призначена для виведення повідомлень у рядок стану, який розташований під заголовком «Постійні». Це означає, що повідомлення не підлягає видаленню, доки служба не буде призупинена або видалена з переднього плану.

Наприклад, музичний програвач, який відтворює музику через цю службу, повинен бути налаштований для роботи на передньому плані, оскільки користувач має повний контроль над його функціоналом. Повідомлення у рядку стану може відображати поточний трек та надавати користувачеві можливість взаємодії з музичним програвачем.

Для запуску служби на передньому плані слід використовувати метод startForeground(). Цей метод приймає два параметри: ціле число, що однозначно ідентифікує повідомлення, та об'єкт Notification для рядка стану. Приклад використання подається в лістингу 1.8 [18].

Лістинг 1.8 – Реалізація методу startForeground()

```
Notification notification = new
Notification(R.drawable.icon,
        getText(R.string.ticker_text),
        System.currentTimeMillis());
Intent notificationIntent = new Intent(this,
ExampleActivity.class);
PendingIntent pendingIntent =
        PendingIntent.getActivity(this,
                0,
                notificationIntent,
                0);
notification.setLatestEventInfo(this,
        getText(R.string.notification_title),
```

```
        getText(R.string.notification_message),
        pendingIntent);
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

1.11 Огляд існуючих рішень

На сьогоднішній день існує порівняно невелика кількість мобільних додатків, які орієнтовані на рішення математичних виразів за допомогою фотографій. Незважаючи на те, що сучасні смартфони стали невід'ємною частиною нашого життя і дозволяють нам здійснювати різноманітні завдання, включаючи навчання та розв'язання математичних задач, програми для розпізнавання та обробки математичних виразів на фотографіях залишаються не такими поширеними, як інші категорії додатків.

Запит «Photomath» в Google Play Market видає лише декілька додатків, які прямо пов'язані з розв'язанням математичних виразів за допомогою фотографій. Хоча ця кількість може здатися обмеженою, варто врахувати, що ці додатки надають можливість користувачам робити математичні обчислення та вирішувати завдання, використовуючи зручний та інтуїтивно зрозумілий спосіб.

У цьому підрозділі буде представлено огляд трьох найпопулярніших і найбільш розповсюджених додатків, які дозволяють вирішувати математичні вирази, зокрема за допомогою фотографій. Кожен з цих додатків має свої особливості та переваги, і ми детально розглянемо їхні функції, можливості та обмеження. При цьому буде надано загальний висновок по кожному додатку, зокрема звертаючи увагу на його зручність, точність розпізнавання та функціональність.

Такий огляд дозволить користувачам зробити обгрунтований вибір щодо вибору програмного рішення для вирішення математичних завдань на платформі Android, а також допоможе виявити основні переваги та недоліки кожного з додатків. Аналіз стане корисним як для учнів та студентів, які шукають засіб для покращення своїх математичних навичок, так і для вчителів та батьків, які цікавляться можливостями використання мобільних додатків у навчанні та розвитку дітей [18].

1.11.1 Photomath

PhotoMath є одним із найбільш популярних додатків для розв'язання математичних виразів за допомогою фотографій. Він розроблений компанією «Photomath, Inc» та надає користувачам можливість легко та швидко вирішувати складні математичні завдання, просто спрямовуючи камеру смартфона на завдання.

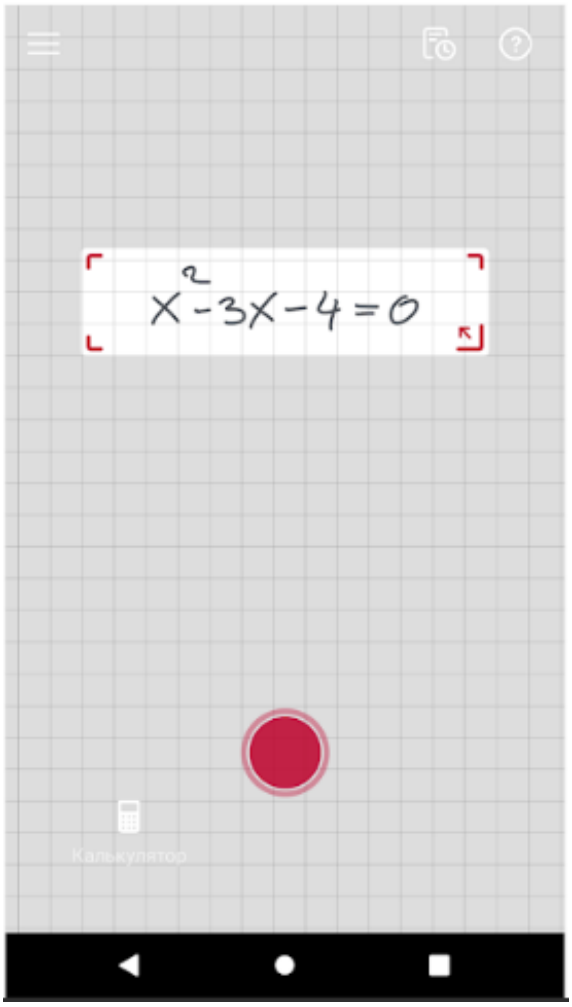

Рисунок 1.15 – Інтерфейс додатку Photomath

Основні переваги PhotoMath:

1) Простий та інтуїтивний інтерфейс: Додаток має дуже зручний та легкий в користуванні інтерфейс. Користувачам потрібно просто сфотографувати математичне завдання, і PhotoMath надасть докладний розв'язок.
2) Широкий спектр підтримуваних завдань: Додаток підтримує різні математичні вирази, включаючи арифметичні операції, лінійні та квадратні рівняння, графіки, тригонометричні функції та інше. Він також вміє вирішувати кроки розв'язку, що корисно для навчання.
3) Спеціальні функції для навчання: PhotoMath містить навчальні ресурси та пояснення для більш глибокого розуміння математичних концепцій. Це допомагає користувачам не лише отримати відповідь, а й дізнатися, як саме вирішувати схожі задачі вручну.
4) Офлайн доступ: PhotoMath може працювати в автономному режимі, що дозволяє користувачам вирішувати завдання навіть без доступу до Інтернету.
5) Підтримка декількох мов: Додаток доступний на багатьох мовах, включаючи українську, що дозволяє користувачам з різних країн використовувати його без перешкод.

Недоліки PhotoMath:

1) Можливість зловживання: PhotoMath може використовуватися для зловживання, зокрема, для копіювання завдань без власних спроб їх розв'язати.
2) Обмежена глибина розуміння: Хоча PhotoMath надає пояснення до завдань, вони можуть бути обмежені за глибиною розуміння. Деякі складні математичні концепції можуть вимагати більш докладного вивчення.
3) Платні функції: Деякі розширені функції PhotoMath є платними, що може обмежити доступність для деяких користувачів.

Загальний висновок:

PhotoMath – це потужний та зручний додаток для розв'язання математичних завдань, який підходить для студентів, вчителів та всіх, хто потребує швидкого та точного розв'язання математичних виразів. Він поєднує в собі простий інтерфейс з можливістю докладного вивчення математичних концепцій. Однак користувачам варто бути обережними щодо можливості зловживання додатком та розглянути обмежену безкоштовну версію перед придбанням розширених функцій.

1.11.2 Mathway: Scan & Solve Problems

Mathway є додатком, призначеним для розв'язання математичних виразів та математичних завдань на платформі Android. Цей додаток вирізняється серед інших аналогів завдяки своєму здатному розв'язувати широкий спектр математичних задач, від базових арифметичних операцій до складних алгебраїчних рівнянь та графіків.

Основні переваги Mathway:

1) Універсальний інструмент для математики: Mathway може розв'язувати різноманітні математичні завдання, включаючи арифметичні операції, алгебру, тригонометрію, геометрію, статистику та інше. Він стане корисним як студентам, так і вчителям, що шукають швидке та точне розв'язання.
2) Докладні кроки розв'язку: Mathway надає докладний опис кроків розв'язку, що допомагає користувачам розуміти, як саме було отримано відповідь. Це особливо корисно для навчання та розвитку математичних навичок.
3) Широкий спектр підтримуваних мов: Додаток працює на різних мовах, включаючи українську, що робить його доступним для користувачів із різних країн.
4) Спеціальний розділ для графіків: Mathway може будувати графіки функцій та показувати їхні точки перегину, нулі та інші характеристики. Це допомагає візуалізувати математичні концепції.
5) Офлайн доступ: Додаток працює в автономному режимі, що дозволяє користувачам розв'язувати завдання навіть без доступу до Інтернету.

Недоліки Mathway:

1) Платний доступ до деяких функцій: Деякі розширені функції та розв'язання конкретних типів завдань можуть бути доступні лише в платній версії додатку, що може відлякувати деяких користувачів.
2) Можливість зловживання: Mathway може використовуватися для копіювання завдань без спроб їх розв'язати самостійно, що не сприяє навчанню.
3) Обмежена безкоштовна версія: Безкоштовна версія Mathway обмежена за функціональністю і не надає доступ до всіх видів математичних завдань.

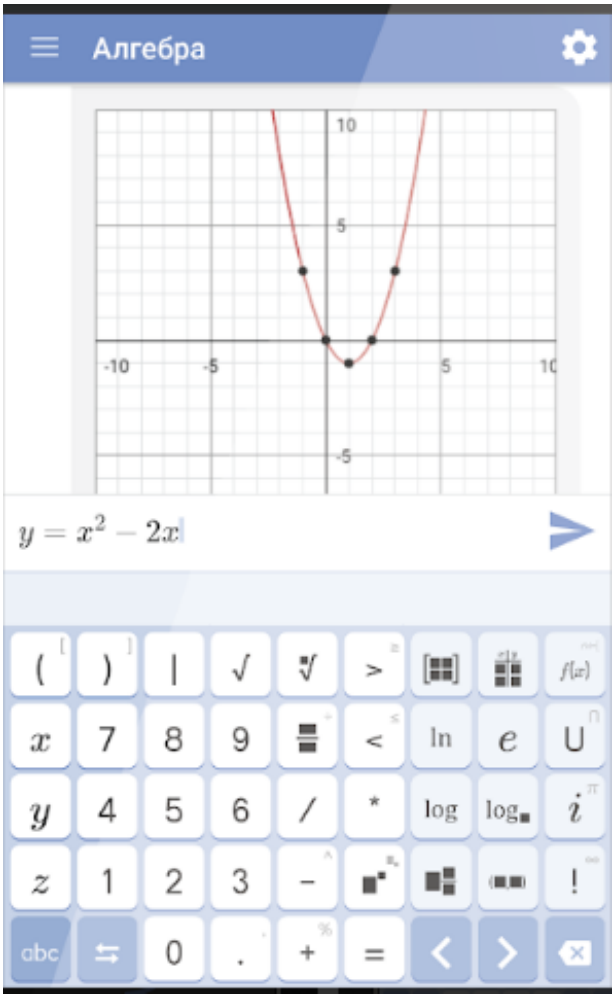

Рисунок 1.16 – Інтерфейс додатку Mathway

Загальний висновок:

Mathway – потужний інструмент для розв'язання математичних завдань, який надає різноманітні функції та підтримує різні мови. Додаток підходить для студентів, вчителів та всіх, хто шукає ефективний інструмент для розв'язання математичних виразів. Однак користувачам слід бути свідомими про можливості зловживання та розглядати обмежену безкоштовну версію перед придбанням розширених функцій.

1.11.3 Microsoft Math Solver

Microsoft Math Solver – це додаток для Android, розроблений корпорацією Microsoft, який допомагає користувачам розв'язувати математичні завдання та вирази за допомогою камери смартфона. Цей додаток відзначається своєю зручністю використання та інтеграцією з іншими продуктами Microsoft, такими як OneNote та Office.

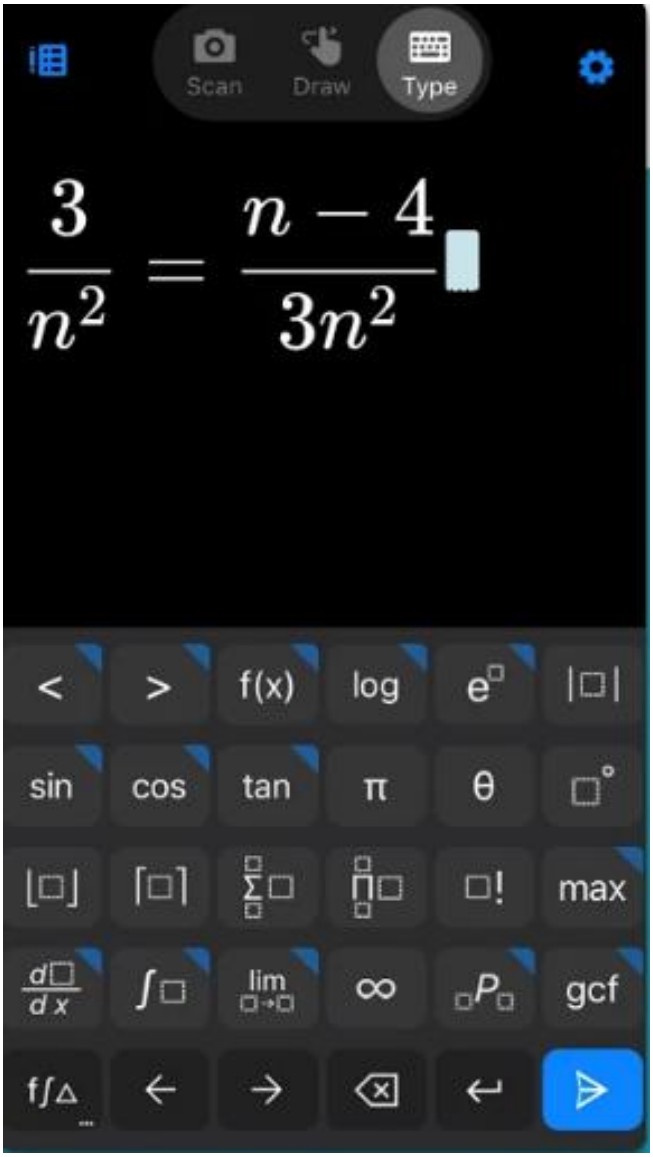

Рисунок 1.17 – Інтерфейс додатку Microsoft Math Solver

Основні переваги Microsoft Math Solver:

1) Спрощена функціональність камери: Додаток дозволяє користувачам просто спрямовувати камеру смартфона на математичне завдання або вираз, а потім автоматично розпізнавати і розв'язувати його. Це особливо зручно для швидкого розв'язання завдань в режимі реального часу.

2) Детальні кроки розв'язку: Microsoft Math Solver надає користувачам докладний опис кроків розв'язку, що допомагає розуміти математичні концепції та методи розв'язку.
3) Інтеграція з OneNote та Office: Додаток може інтегруватися з іншими продуктами Microsoft, такими як OneNote та Office. Це означає, що користувачі можуть легко імпортувати математичні завдання та розв'язки до своїх навчальних матеріалів та документів.
4) Офлайн доступ: Microsoft Math Solver працює в автономному режимі, що дозволяє користувачам розв'язувати завдання навіть без доступу до Інтернету.

Недоліки Microsoft Math Solver:

1) Можливість зловживання: Як і в інших подібних додатках, Microsoft Math Solver може використовуватися для копіювання завдань без спроб їх розв'язати самостійно. Це може бути проблемою для навчання та розвитку навичок.
2) Обмежена безкоштовна версія: Деякі розширені функції та інструменти розв'язку завдань доступні лише в платній версії додатку, що може відлякувати деяких користувачів.

Загальний висновок:

Microsoft Math Solver – це потужний та зручний додаток для розв'язання математичних завдань та виразів за допомогою камери смартфона. Він ідеально підходить для школярів, студентів та всіх, хто шукає ефективний інструмент для розв'язання математичних завдань. Однак користувачам слід бути свідомими можливості зловживання та розглядати обмежену безкоштовну версію перед придбанням розширених функцій.

1.12 Постановка завдання та вибір методів реалізації

Для успішної реалізації нашого додатку, спершу нам потрібно визначити середовище розробки, у якому ми будемо працювати. Для створення додатків на платформі Android існують різні середовища програмування, але рекомендованим та найбільш популярним варіантом є використання Android Studio. Android Studio спеціально створена для розробки під операційну систему Android і забезпечує розробникам широкий функціонал та зручність у створенні додатків. Важливо відзначити, що Android Studio базується на програмному забезпеченні IntelliJ IDEA від компанії JetBrains, що гарантує високий рівень продуктивності та можливостей для розробників.

Спершу, для реалізації додатку, який дозволить користувачам розв'язувати математичні вирази на основі фотографій, було ретельно проаналізовано функціональні можливості існуючих додатків, які пропонують аналогічний функціонал. Це допомогло сформувати чіткий перелік функцій, які планується внести до створюваного додатку.

1) Сканування друкованих математичних завдань.

Для цього завдання планується використовувати ML Kit – мобільний набір інструментів для роботи з машинним навчанням від Google. ML Kit дозволить розпізнавати текст на фотографіях та друкувати математичні вирази для подальшого розв'язку. Основною перевагою ML Kit є можливість обробки інформації без необхідності використовувати зовнішні сервіси чи API, що забезпечує швидкість та надійність розпізнавання.

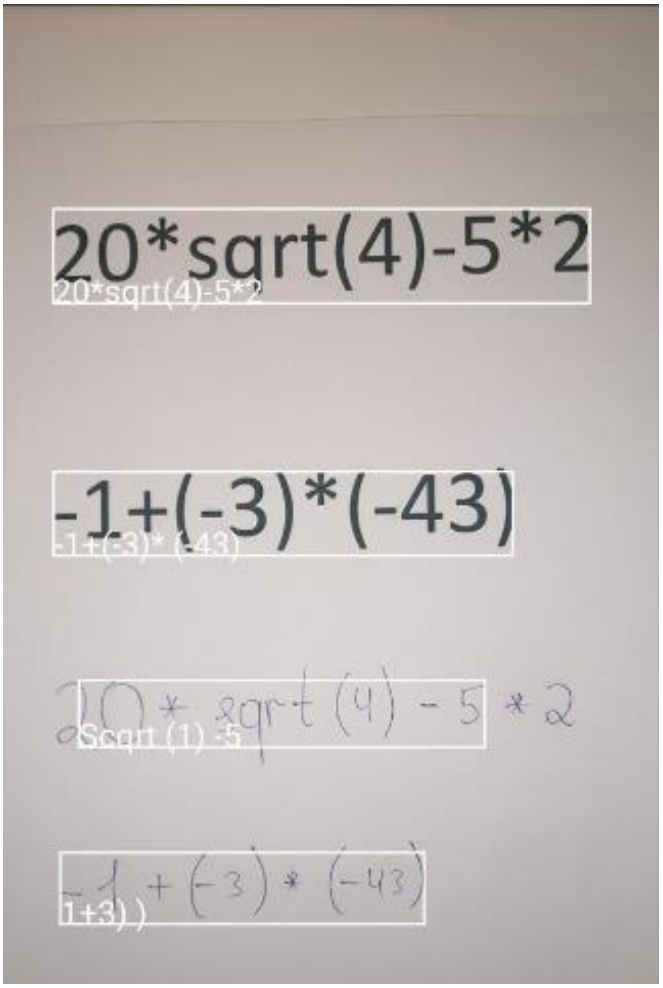

Рисунок 1.18 – Перший тест сканування виразів

2) Розв'язання відсканованих математичних завдань за допомогою WolframAlpha.

Додавши підтримку WolframAlpha до Photomath, ми можемо розширити можливості додатку та надати користувачам доступ до більш розгорнутих математичних розв'язань.

WolframAlpha – це веб-сервіс, який надає широкий спектр математичних, наукових та інженерних розрахунків. Інтеграція з цим сервісом дозволить Photomath використовувати потужність обчислень WolframAlpha для розв'язання складних математичних задач та виразів.

Переваги інтеграції з WolframAlpha:

1. Розширені можливості розв'язання. Інтеграція з WolframAlpha дозволить користувачам Photomath отримувати більш докладні та розгорнуті розв'язання для складних математичних завдань.
2. Навчальний контекст. Використання WolframAlpha допоможе створити навчальний контекст для користувачів, які можуть докладніше ознайомитися зі способами розв'язання математичних завдань.
3. Спрощення процесу навчання. Інтеграція з WolframAlpha сприятиме покращенню якості освіти та зрозумінню матеріалу учнями.

3) Взаємодія з користувачем.

Додаток має надавати зручний та інтуїтивний інтерфейс для користувача, який дозволить легко завантажувати фотографії, перевіряти результати, а також отримувати пояснення до кожного кроку розв'язання задачі.

4) Пояснення розв'язання.

Додаток повинен надавати користувачам не лише результат розв'язання математичних виразів, але і пояснення кожного кроку процесу розв'язання. Це допоможе користувачам розуміти математичні концепції та покращити їхні навички.

5) Можливість зберігати та підтримувати історію завдань.

Для зручності користувачів, буде розглянуто можливість створення історії завдань, яку користувачі зможуть переглядати, повторювати та використовувати для навчання. Історія завдань також допоможе користувачам відстежувати їхні покращення та прогрес у вирішенні математичних завдань.

Після визначення функціональності додатку та вибору середовища розробки Android Studio, планується реалізувати основні етапи розробки:

1. Розробка інтерфейсу користувача.
   Перший крок полягатиме в створенні зручного та інтуїтивно зрозумілого інтерфейсу для користувача. Це включає в себе створення екранів для завантаження фотографій, відображення результатів, пояснень та історії завдань.
2. Розробка сканера математичних виразів.
   Другим кроком буде створення сканера, який зможе розпізнавати математичні вирази на фотографіях та перетворювати їх на текстовий вигляд для подальшого розв'язання. Для цього буде використано ML Kit від Google.
3. Розробка алгоритму розв'язання математичних виразів.
   Третім етапом буде створення алгоритму, який зможе вирішувати математичні вирази. Цей алгоритм буде використовувати вбудовані функції мови програмування Java для виконання операцій над числами.
4. Розробка системи пояснень.
   Четвертим етапом буде створення системи пояснень, яка буде генерувати пояснення до кожного кроку розв'язання математичного

виразу. Буде розглянуто можливості інтеграції з відкритими математичними бібліотеками для додавання роз'яснень.

5. Розробка системи зберігання історії завдань.
   Останнім етапом буде створення системи зберігання історії завдань, яка дозволить користувачам переглядати, повторювати та використовувати розв'язані завдання для навчання.

Висновки за розділом

В першому розділі кваліфікаційної роботи було розглянуто основні відомості про Android та Java та проаналізовано існуючі програмні засоби та моделі для розв’язання математичних задач.

2 РОЗРОБКА МОБІЛЬНОГО ДОДАТКА PHOTOMATH

2.1 Реалізація методів основного функціоналу додатка

2.1.1 Налаштування основної камери та розпізнавання тексту

Інтеграція функцій основної камери та розпізнавання тексту є ключовим аспектом розробки сучасних мобільних додатків, що спрямовані на полегшення вивчення та розуміння математичних концепцій.

Для реалізації цих функцій був створений метод createCameraSource, код якого представлений в лістингу 2.1, в якому було використано стандартні android java класи CameraSource та TextRecognizer (обробляє зображення і визначає, який текст з'явився всередині).

Лістинг 2.1 – Створення камери

```
private void createCameraSource(boolean autoFocus,
boolean useFlash) {
    Context context = getApplicationContext();
    TextRecognizer textRecognizer = new
TextRecognizer.Builder(context).build();
    textRecognizer.setProcessor(new
OcrDetectorProcessor(graphicOverlay));

    //Перевірка на достатню кількість пам'яті для
роботи TextRecognizer
    if (!textRecognizer.isOperational()) {
        IntentFilter lowstorageFilter = new
IntentFilter(Intent.ACTION_DEVICE_STORAGE_LOW);
        boolean hasLowStorage = registerReceiver(null,
lowstorageFilter) != null;
        if (hasLowStorage) {
            Toast.makeText(this,
R.string.low_storage_error, Toast.LENGTH_LONG).show();
        }
    }
    cameraSource =
            new
CameraSource.Builder(getApplicationContext(),
textRecognizer)
```

```
.setFacing(CameraSource.CAMERA_FACING_BACK)
                    .setRequestedPreviewSize(1280,
1024)
                    .setRequestedFps(2.0f)
                    .setFlashMode(useFlash ?
Camera.Parameters.FLASH_MODE_TORCH : null)
                    .setFocusMode(autoFocus ?
Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO : null)
                    .build();
}
```

Android надає розробникам потужні інструменти для взаємодії з камерою та роботи з розпізнаванням тексту, що відкриває безліч можливостей для створення функціональних та інноваційних додатків. Важливими компонентами цього процесу є класи CameraSource та TextRecognizer, які дозволяють взаємодіяти з камерою та використовувати сервіс розпізнавання тексту.

1. CameraSource:

Створення камери: CameraSource – це клас, який відповідає за управління основною камерою пристрою.

Визначення параметрів камери: розробник може визначити такі параметри, як розмір попереднього перегляду, кадрова частота, обране напрямок (передня або задня камера), режим спалаху та режим фокусування.

2. TextRecognizer:

Створення розпізнавача тексту: TextRecognizer дозволяє виявляти та розпізнавати текст на зображеннях, забезпечуючи можливість витягувати значущий текст з оточуючого середовища.

Обробка результатів розпізнавання: розпізнавач тексту дозволяє встановлювати спеціальний обробник (процесор), який опрацьовує результати розпізнавання, як, наприклад, визначений в коді OcrDetectorProcessor.

3. Інтеграція з Android-додатком:

Використання контексту додатка: Отримання контексту додатка є необхідною частиною налаштування камери та розпізнавання тексту.

Оптимізація роботи з пам'яттю: Перевірка на операбельність TextRecognizer та опціональне повідомлення про низький рівень вільної пам'яті дозволяє покращити користувацький досвід та уникнути проблем при роботі додатка.

4. Додаткові можливості:

Динамічне налаштування параметрів камери: в коді можливо встановлювати параметри камери залежно від умов використання, таких як фокус, спалах та інші.

Повідомлення про помилки та стан камери: Виведення повідомлень про помилки або низький рівень вільної пам'яті дозволяє покращити стабільність та надійність додатка.

Для того, щоб зчитувати текст прямо під час роботи камери, а не тільки на окремих кадрах, як працює метод виявлення в TextRecognizer, було реалізовано інтерфейс Detector.Processor (лістинг 2.2), який обробляє текст, як тільки він з’явиться на екрані.

Лістинг 2.2 – DetectorProcessor

```
public class OcrDetectorProcessor implements
Detector.Processor<TextBlock> {
    private GraphicOverlay<OcrGraphic> graphicOverlay;
    OcrDetectorProcessor(GraphicOverlay<OcrGraphic>
ocrGraphicOverlay) {
        graphicOverlay = ocrGraphicOverlay;
    }
    //отримуємо всі TextBlocks і створюємо об'єкти
OcrGraphic для кожного текстового блоку, виявленого
процесором
    @Override
    public void
receiveDetections(Detector.Detections<TextBlock>
detections) {
        graphicOverlay.clear();
        SparseArray<TextBlock> items =
detections.getDetectedItems();
        for (int i = 0; i < items.size(); ++i) {
            TextBlock item = items.valueAt(i);
            if (item != null && item.getValue() !=
null) {
                OcrGraphic graphic = new
OcrGraphic(graphicOverlay, item);
                graphicOverlay.add(graphic);
            }
        }
    }
    @Override
    public void release() {
        //звільнення від ресурсів при знищенні
TextRecognizer
        graphicOverlay.clear();
    }
}
```

Після створення процесора можна вибрати саме ті текстові блоки які знаходяться в межах червоних ліній. Для цього було пройдено по всіх текстових блоках, які з'явились на канвасі (лістинг 2.3).

Лістинг 2.3 – Визначення потрібного тексту

```
List<? extends Text> textComponents =
textBlock.getComponents();
    for(Text currentText : textComponents) {
    float bottom =
translateY(currentText.getBoundingBox().bottom);
    if(bottom > 600 && bottom < 750){
        OcrCaptureActivity.textExpr =
currentText.getValue();
    }
}
```

Також був створений метод для надання користувачем доступу до камери (лістинг 2.4).

Лістинг 2.4 – Доступ до камери

```
private void requestCameraPermission() {

    final String[] permissions = new
String[]{Manifest.permission.CAMERA};

    if
(!ActivityCompat.shouldShowRequestPermissionRationale(t
his,
            Manifest.permission.CAMERA)) {
        ActivityCompat.requestPermissions(this,
permissions, RC_HANDLE_CAMERA_PERM);
        return;
    }

    final Activity thisActivity = this;

    View.OnClickListener listener = new
View.OnClickListener() {
        @Override
        public void onClick(View view) {

ActivityCompat.requestPermissions(thisActivity,
permissions,
                    RC_HANDLE_CAMERA_PERM);
        }
```

```
    };

    Snackbar.make(graphicOverlay,
R.string.permission_camera_rationale,
            Snackbar.LENGTH_INDEFINITE)
            .setAction(R.string.ok, listener)
            .show();
}
```

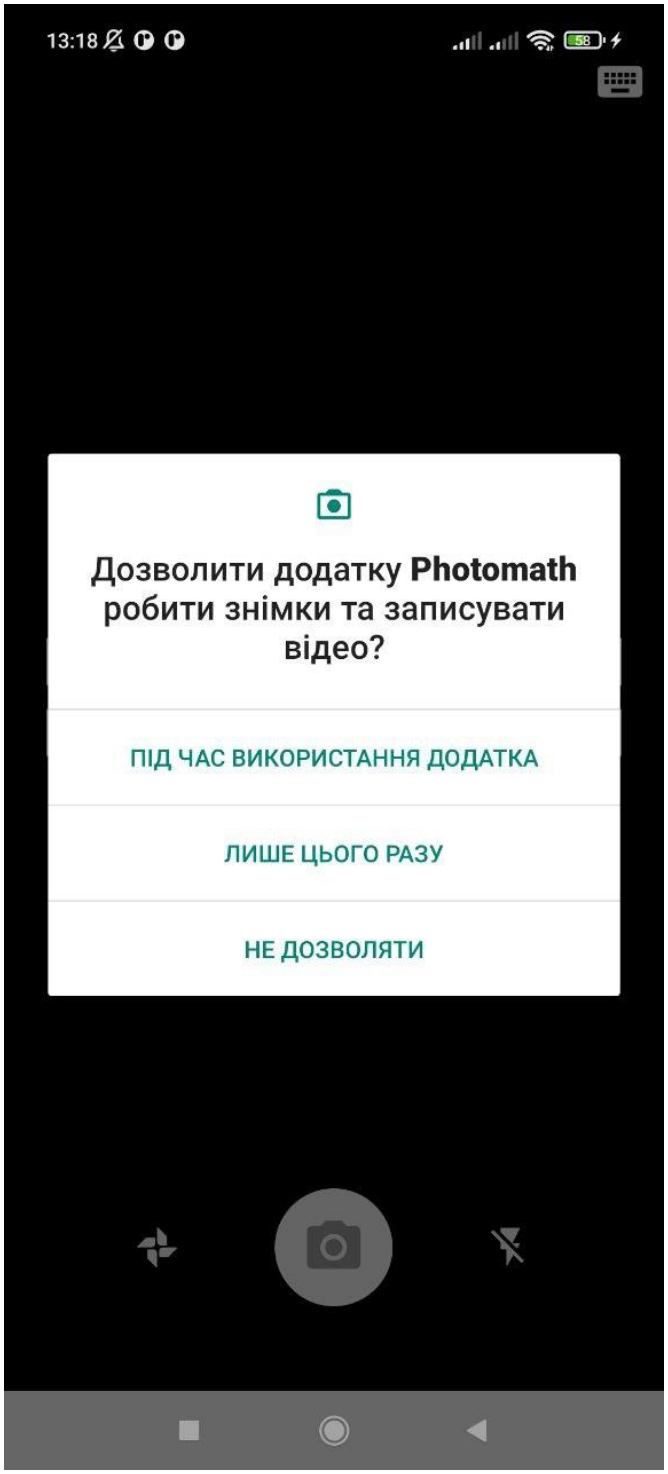

Рисунок 2.1 – Доступ до камери

2.1.2 Створення калькулятора

Калькулятор був створений на основі Wolfram Alpha, що надає доступ до потужного обчислювального сервісу та широкого спектру математичних функцій і інформації.

Wolfram Alpha – це велика обчислювальна машина та пошуковий двигун, який надає результати з різних областей, включаючи математику, фізику, хімію, біологію, географію та інші.

Основні характеристики та функції:

A. Синтаксичне введення. Взаємодія з Wolfram Alpha може відбуватися через текстовий ввід, де користувачі можуть задавати питання або виражати математичні вирази у природній мові.
B. Розпізнавання математичних виразів. Wolfram Alpha вміє розпізнавати та обчислювати різноманітні математичні вирази, включаючи арифметичні операції, алгебру, інтеграли, диференціали та багато іншого.
C. Графіки та візуалізація. Підтримка виведення графіків функцій, рівнянь та інших математичних об'єктів для зручного візуального аналізу результатів.
D. Різноманітні області знань. Поза математикою, Wolfram Alpha може надавати інформацію з різних галузей, таких як фізика, хімія, біологія, географія, історія та інші, що розширює його потенціал для використання у навчанні та дослідженнях.
E. API для розробників. Wolfram Alpha надає API, що дозволяє розробникам інтегрувати його функціонал у свої додатки, створюючи, таким чином, власні калькулятори або інші обчислювальні інструменти.

Створення калькулятора:

1. Отримання API-ключа.

Отримання API-ключа для використання функціоналу Wolfram Alpha є першим та необхідним кроком для інтеграції цього потужного обчислювального сервісу у власний додаток. Цей процес забезпечує розробників доступом до розширених можливостей математичного обчислення, аналізу та отримання інформації з різних галузей.

Починаючи, розробники повинні завітати на офіційний веб-сайт Wolfram Alpha та визначити розділ, присвячений розробникам і API. У цьому розділі зазвичай надані важливі відомості та інструкції щодо отримання API-ключа.

Під час реєстрації на веб-сайті Wolfram Alpha для отримання API-ключа, розробники можуть створити власний обліковий запис, де надають особисті дані та інформацію про своє програмне забезпечення. У цьому контексті, Wolfram Alpha може вимагати вказати призначення використання API, об'єм запитань, які планується виконати, та інші параметри, які допомагають налагоджувати взаємодію з сервісом.

Після успішної реєстрації, розробники отримують унікальний API-ключ, який стає ключовим елементом для взаємодії їхнього додатку з Wolfram Alpha. Цей ключ дозволяє системі впізнавати та авторизувати запити, які надходять від конкретного додатку, забезпечуючи при цьому безпеку та відстеження використання сервісу.

Отримавши API-ключ, розробники можуть використовувати його в їхньому програмному коді для взаємодії з Wolfram Alpha API, використовуючи потужні можливості цього сервісу у своєму додатку.

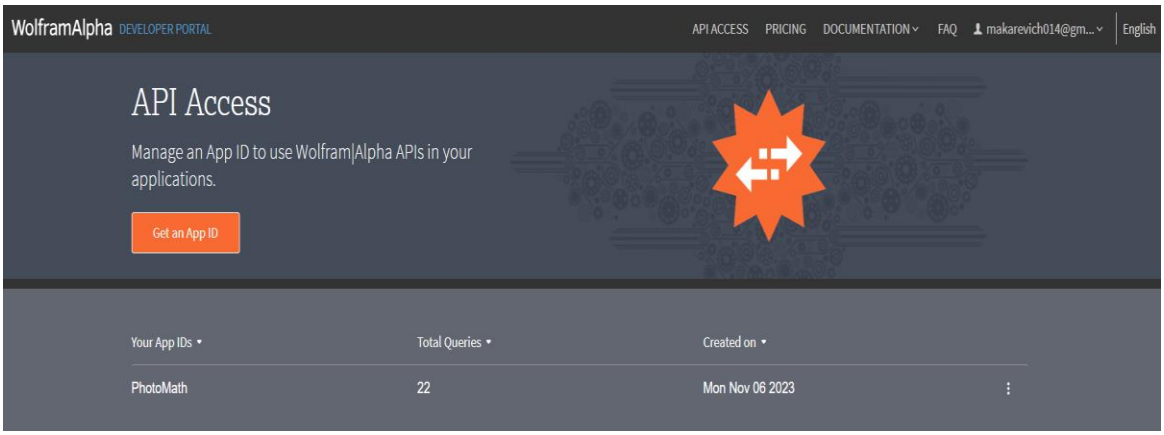

Рисунок 2.2 – Wolfram Alpha API Access

2. Інтеграція API.

Інтеграція Wolfram Alpha API в розробку калькулятора включає створення класу, який спрощує комунікацію між додатком та обчислювальним сервісом. Цей процес дозволяє використовувати потужні можливості Wolfram Alpha для вирішення різноманітних математичних завдань та отримання детальної інформації.

Розробка класу калькулятора, що взаємодіє з Wolfram Alpha API, розпочинається включенням необхідних бібліотек або пакетів, які дозволяють здійснювати HTTP-запити та обробляти відповіді сервісу. Розробники можуть використовувати мови програмування, такі як Java, Python, або інші, залежно від вибору платформи для розробки.

Після імпорту необхідних інструментів, розробники створюють клас, який містить логіку взаємодії з Wolfram Alpha API. Цей клас може включати методи для формування та відправлення HTTP-запитів до сервісу, передачі математичних виразів для обчислення, а також обробки та аналізу відповідей.

Ключовим елементом цього класу є методи, які передають запити до Wolfram Alpha та обробляють отримані результати (лістинг 2.5, лістинг 2.6). Розробники повинні дотримуватися документації Wolfram Alpha API, щоб правильно форматувати запити та обробляти відповіді.

Після визначення методів обміну інформацією з API, розробники можуть інтегрувати цей клас у свій калькулятор, дозволяючи користувачам використовувати розширені математичні функції Wolfram Alpha безпосередньо в додатку.

Окрім базової функціональності калькулятора, інтеграція API дозволяє розробникам використовувати інші можливості Wolfram Alpha, такі як отримання графіків, розв'язання складних математичних рівнянь чи надання контекстної інформації з різних наукових галузей. Все це розширює функціонал калькулятора та робить його потужним інструментом для вивчення та розвитку математичних навичок.

Лістинг 2.5 – Створення та передача запиту до Wolfram Alpha

```
private static final String BASE_URL =
"https://api.wolframalpha.com/v2/query?input=";
private static final String FORMAT =
"&format=image,plaintext";
private static final String OUTPUT = "&output=JSON";
private static final String APP_ID = "&appid=" +
APIKey.ID;

WolframAlphaAsyncTask wolframTask = new
WolframAlphaAsyncTask(adapter,
getApplicationContext());
String input = Converter.convertMathToWolfram(result);
input = input.trim();
String urlString = createFullURL(input);
Log.e("URL:", urlString);
wolframTask.execute(urlString);
```

Лістинг 2.6 – Отримання результату запиту від Wolfram Alpha

```
protected WolframResult doInBackground(String...
params) {

    String urlString = params[0];
    Log.e("URL?", urlString);

    try {
        URL url = new URL(urlString);
        InputStream inStream = url.openStream();
        InputStreamReader reader = new
InputStreamReader(inStream, Charset.forName("UTF-8"));
        JsonReader jsonReader = new JsonReader(reader);
        Gson gson = new Gson();
        WolframResult solutionsList =
gson.fromJson(jsonReader, WolframResult.class);
        return solutionsList;
    } catch (MalformedURLException badURL) {
        badURL.printStackTrace();
        throw new RuntimeException();
```

```
    } catch (IOException networkError) {
        networkError.printStackTrace();
        return null;}
}
```

3. Візуалізація результатів:

Візуалізація результатів обчислень грає ключову роль у поліпшенні користувацького досвіду використання калькулятора, і особливо коли це пов'язано з інтеграцією з потужними сервісами, такими як Wolfram Alpha. Відображення результатів у формі графіків або інших візуальних елементів дозволяє користувачам краще розуміти та аналізувати математичні концепції та дані.

Після того, як калькулятор отримує результати від Wolfram Alpha API, наступним кроком є їхнє візуальне представлення. Це може бути досягнуто шляхом використання вбудованих бібліотек для графіків або інших інструментів для візуалізації даних, які надає обрана платформа розробки.

Зазвичай, розробники створюють модуль візуалізації, який приймає числові дані та конвертує їх у візуальні елементи, зрозумілі для користувача. Якщо результати включають графіки функцій або залежності, цей модуль може генерувати відповідні графіки, які надають графічне представлення обчисленим даним.

Крім того, можливе використання інших форм візуалізації, таких як діаграми, кругові графіки або інші графічні елементи, що допомагають в інтуїтивному сприйнятті складних математичних концепцій чи результатів.

Цей етап важливий для полегшення взаємодії з калькулятором та впровадження додаткового рівня зрозумілості. Візуалізація робить математичні концепції доступнішими і сприяє розвитку математичної інтуїції користувачів, роблячи процес обчислень більш динамічним та залучаючим.

Лістинг 2.7 – Візуалізація результатів запиту

```
Subpod currentSubpod = results.get(position);
if(pods.length > 0){
    for(Pod pod : pods){
        for(Subpod subpod : pod.getSubpods()){
            if(subpod.equals(currentSubpod)){

holder.itemLabel.setText(pod.getTitle());

holder.plainText.setText(results.get(position).getPlain
Text());
                }
            }
    }
}
```

```
if(targets.length > 0){

    String itemLabel =
holder.itemLabel.getText().toString();
    if(itemLabel.equals("Result")
            || itemLabel.equals("Input")
            || itemLabel.equals("Number name")
            || itemLabel.equals("Typical human
computation times")
            || itemLabel.equals("Geometric figure")
            || itemLabel.equals("Properties as a
function")
            || itemLabel.equals("Properties as a real
function")
            || itemLabel.equals("Equation")
            || itemLabel.equals("Input interpretation")
            || itemLabel.equals("Alternate forms")
            || itemLabel.equals("Alternate form")
            || itemLabel.equals("Solution")
            || itemLabel.equals("Continued fraction")
            || itemLabel.equals("Total")
            || itemLabel.equals("Midpoint")
            || itemLabel.contains("Line through")
            || itemLabel.contains("Exact result")
            || itemLabel.contains("Decimal")
            || itemLabel.contains("Unit conversions")
            || itemLabel.contains("Interpretations")
            || itemLabel.contains("Basic unit
dimensions")){

holder.solutionImage.setVisibility(View.GONE);

holder.plainText.setVisibility(View.VISIBLE);
    } else {

holder.solutionImage.setVisibility(View.VISIBLE);
        holder.plainText.setVisibility(View.GONE);
        String currentURL =
results.get(position).getImage().getSourceLink();
Picasso.with(context).load(currentURL).into(holder.solu
tionImage);
    }
}
```

4. Оптимізація та вдосконалення:

Оптимізація та вдосконалення калькулятора, який використовує Wolfram Alpha API, є важливим етапом для забезпечення ефективності та задоволення користувачів. Цей процес включає у себе ряд заходів, які спрямовані на поліпшення швидкодії, розширення функціоналу та вдосконалення інтерфейсу.

Оптимізація швидкодії включає в себе аналіз та вдосконалення ефективності алгоритмів, які використовуються для обчислень та взаємодії з Wolfram Alpha API. Розробники можуть оптимізувати процеси обробки запитань, використовуючи кращі методи або кешування деяких результатів для швидшого доступу. Це дозволяє калькулятору реагувати на запитання користувачів інтуїтивно та без зайвого затримання.

Додавання додаткових функцій – це розширення функціоналу калькулятора, щоб задовольнити різноманітні потреби користувачів. Це може включати нові математичні операції, можливості взаємодії з іншими обчислювальними сервісами чи навіть додаткові можливості візуалізації результатів. Додаткові функції розширюють можливості калькулятора та роблять його більш універсальним і потужним інструментом.

Вдосконалення інтерфейсу включає в себе поліпшення користувацького досвіду через зручніший та інтуїтивно зрозумілий дизайн. Це може включати оптимізацію розташування елементів, додавання інтерактивності, вдосконалення візуального оформлення та забезпечення простоти використання. Покращений інтерфейс забезпечує зручність взаємодії користувачів з додатком, роблячи його більш привабливим та доступним.

Оптимізація та вдосконалення є невід’ємною частиною постійного розвитку калькулятора, щоб відповідати зростаючим вимогам користувачів та забезпечити ефективне та задовільне використання математичних обчислень.

2.1.2 Створення Activity

Створення Activity у мобільному додатку на платформі Android є ключовим етапом в розробці користувацького інтерфейсу та функціоналу. Activity представляє екран або вікно, яке взаємодіє з користувачем, і є однією з основних компонентів Android-додатків.

Процес створення Activity зазвичай починається з ознайомлення розробників зі специфікаціями та вимогами для даного екрану. Розробники визначають, які компоненти і функціонал будуть включені, які дані вони будуть відображати та які події будуть обробляти. Це визначається в файлі розмітки XML, де описується вигляд Activity.

Далі, розробники створюють відповідний клас Java, який розширює клас Activity. У цьому класі визначаються методи життєвого циклу, такі як

onCreate(), onPause(), onResume() і багато інших, які дозволяють керувати різними етапами життєвого циклу Activity.

У методі onCreate() виконується ініціалізація Activity, встановлюються розмітка, обробники подій, ініціалізуються об'єкти та налаштовуються параметри. Після цього, при потребі, може проводитися взаємодія з іншими компонентами додатку, такими як сервіси, бази даних чи інші Activity.

Життєвий цикл Activity включає в себе різні етапи, такі як створення, запуск, пауза, відновлення та знищення. Кожен з цих етапів має свої особливості, і розробники можуть використовувати їх для ефективного управління ресурсами та станом додатку.

Створення основних Activity у додатку починається з визначення екранів, які будуть першими відображатися при запуску додатку. Першою сторінкою, яку зазвичай розробляють, є сторінка завантаження. Ця сторінка відповідає за вітання користувача та ініціалізацію необхідних ресурсів.

Процес створення сторінки завантаження розпочинається з визначення вигляду цієї Activity у файлі розмітки XML. Тут розробники можуть визначити елементи інтерфейсу, такі як зображення, текстові поля чи інші компоненти, які відображатимуться на екрані завантаження.

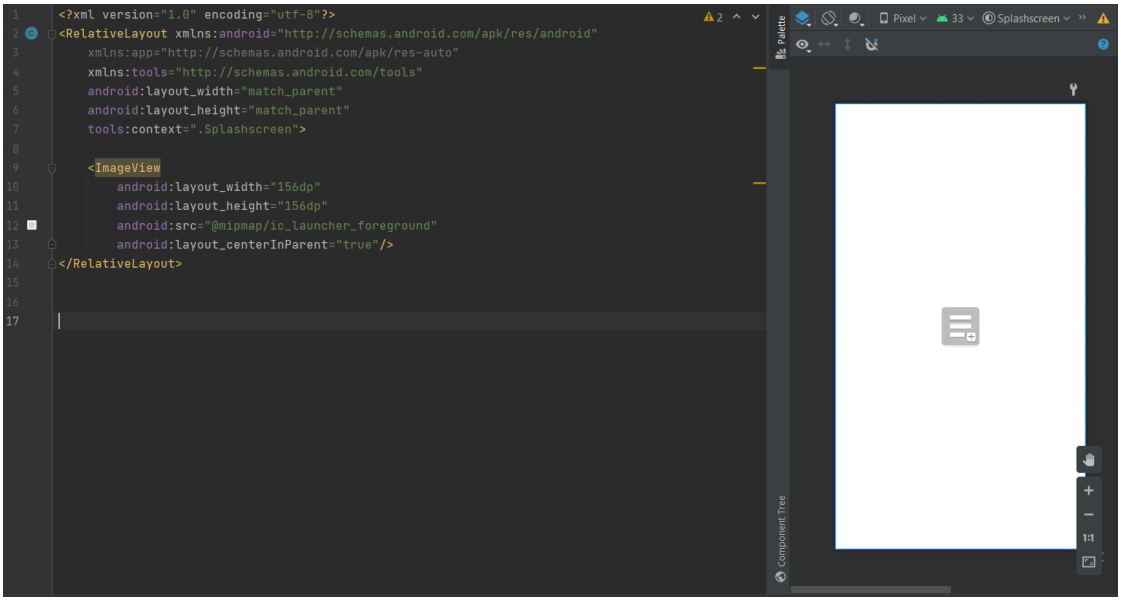

Рисунок 2.3 – Loading Activity

Після визначення розмітки, вони переходять до створення відповідної Java-класу для Activity. У цьому класі розробники реалізовують логіку завантаження, яка може включати в себе ініціалізацію даних, перевірку підключення до мережі чи інші дії, які варто виконати перед відображенням основної частини додатку.

Крім того, розробники можуть використовувати сторінку завантаження для реалізації анімацій, які додають динамічність та привабливість до інтерфейсу. Це може бути корисно для створення позитивного першого враження від додатку.

Щоб забезпечити коректну роботу додатку, розробники також можуть використовувати сторінку завантаження для перевірки необхідних дозволів чи налаштувань пристрою, таких як доступ до камери, геолокації чи інших ресурсів.

Після успішного завантаження сторінки завантаження перед користувачем відображається основна сторінка додатку. Цей етап є ключовим, оскільки саме тут користувач здійснює основні взаємодії з додатком.

Процес створення основної сторінки починається з визначення її вигляду в файлі розмітки XML. Тут розробники визначають компоненти інтерфейсу, такі як кнопки, тексти, зображення, а також визначають їхнє розташування та взаємодію.

У відповідному Java-класі, який реалізує основну сторінку, відбувається ініціалізація компонентів, встановлення обробників подій та реалізація логіки, яка відповідає за взаємодію з користувачем. Тут розробники можуть реалізовувати різноманітні функції, такі як відображення даних з бази даних, виклик інших Activity, чи інші операції, які забезпечують основний функціонал додатку.

Основна сторінка може також включати в себе вивчення стану додатку та його компонентів для забезпечення правильної відповіді на взаємодію користувача. Розробники можуть використовувати певні оптимізації для підвищення продуктивності та ефективності додатку на цьому етапі.

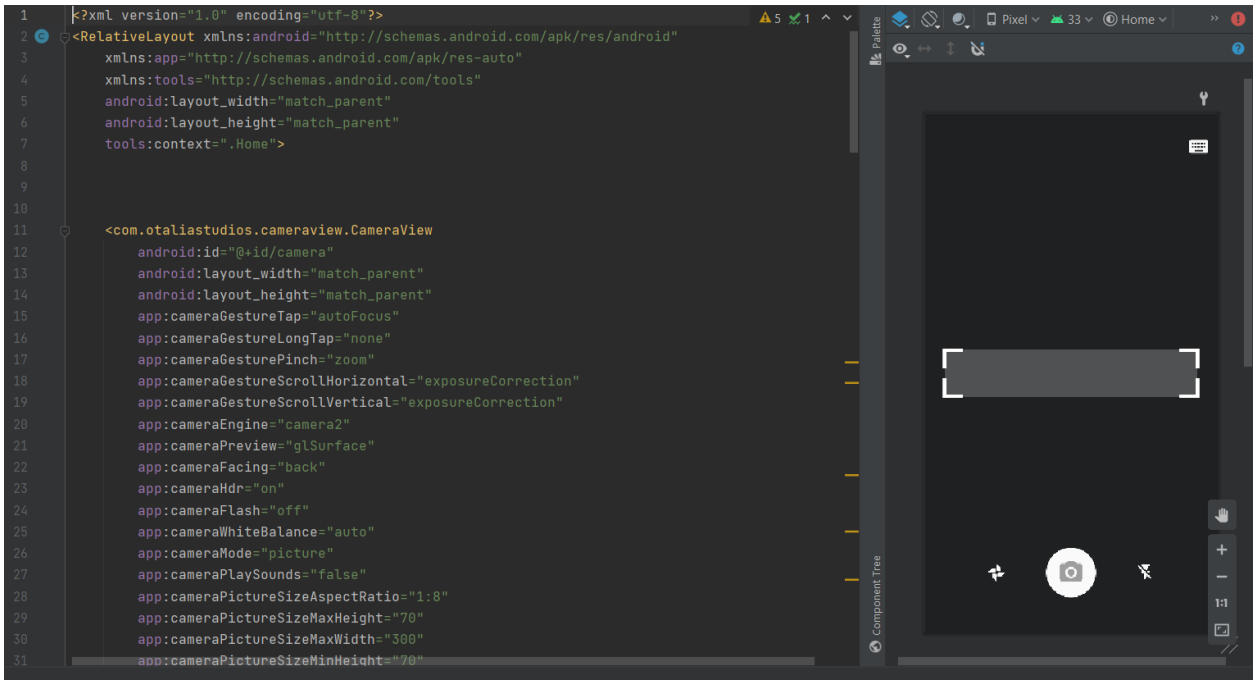

Рисунок 2.4 – Main Activity

Після того, як користувач натискає на зображення на основній сторінці додатку, викликається діалогове вікно, яке надає додаткові опції або виводить деталізовану інформацію. Процес створення цього діалогового

вікна та сторінки для представлення детального рішення включає кілька ключових етапів.

Розробники визначають вигляд та функціонал діалогового вікна в файлі розмітки XML, визначаючи компоненти інтерфейсу та їх взаємодію. Діалогове вікно може містити кнопки для вибору опцій, текстові поля чи інші елементи, які дозволяють користувачеві взаємодіяти з додатком на більш глибокому рівні.

Після визначення розмітки, розробники переходять до реалізації логіки діалогового вікна в Java-класі. Тут вони встановлюють обробники подій для взаємодії з введеними користувачем даними та реалізують логіку відображення або приховання деяких елементів в залежності від обраної опції.

Спеціальна сторінка для представлення детального рішення визначається аналогічно: встановлення розмітки та логіки відображення необхідної інформації. Тут розробники можуть використовувати різні елементи, такі як текстові поля, зображення чи графіки, для надання детального огляду інформації, пов'язаної з обраною опцією.

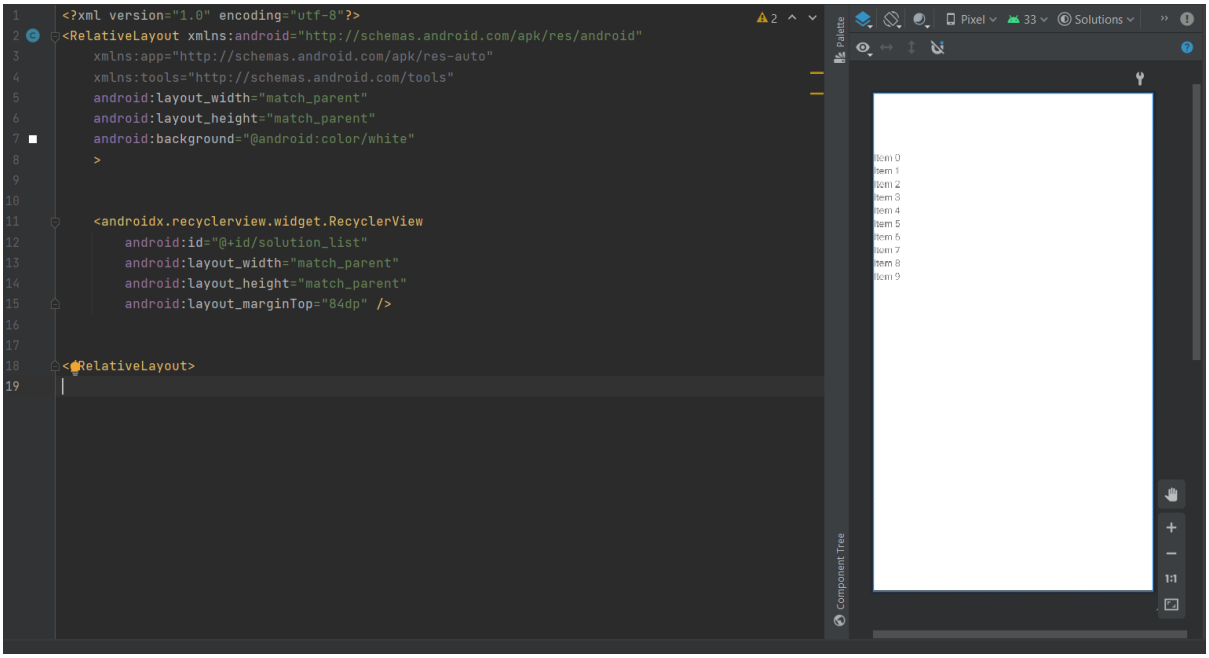

Рисунок 2.5 – Solution Activity

Висновки за розділом

В другому розділі кваліфікаційної роботи було реалізовано основні методи для роботи додатка та створено основні Activity.

3 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКА РНОТОМАТН

3.1 Поняття якості програмного забезпечення

Якість представляє собою міру відповідності системи, компонента чи процесу визначеним вимогам, потребам або очікуванням користувача. Для оцінки ефективності системи, компонента або процесу застосовують атрибути якості – характеристики, що відображають дану властивість. Поняття якісного програмного забезпечення може бути зрозумілим, якщо запитати різноманітну групу фахівців у розробці, продажу та використанні програмного забезпечення. Отримані відповіді об'єднуються навколо наступних критеріїв:

- легкість використання;
- висока продуктивність;
- відсутність помилок;
- можливість використання на різних платформах;
- працездатність 24/7;
- легкість розширення новими функціями;
- відповідність потребам користувачів.

Ці критерії визначають характеристики, які є ключовими для окремого користувача, розробника програмного забезпечення або групи фахівців. Однак для максимального задоволення всіх користувачів програмного забезпечення, досягнення стійкої позиції на ринку та розширення потенціалу розвитку, важливо враховувати всі зазначені характеристики. Таким чином, якість програмного забезпечення може бути охарактеризована широким спектром різноманітних характеристик [19].

Характеристики програмного забезпечення поділяються на декілька важливих аспектів, що визначають його якість:

1. Функціональність (Functionality):
   - спроможність програмного забезпечення ефективно вирішувати завдання, що відповідають вимогам і потребам користувача в умовах, визначених для його використання;
   - забезпечення надійної, точної та сумісної роботи, відповідність стандартам галузі та захист від несанкціонованого доступу.
2. Надійність (Reliability):
   - здатність програмного забезпечення виконувати потрібні завдання визначеним чином протягом заданого періоду часу або кількості операцій;

- завершеність та цілісність системи, самостійне та коректне відновлення після збоїв та відмов.

3. Зручність використання (Usability):
   - легкість розуміння, вивчення та використання програмного забезпечення користувачами.
4. Ефективність (Efficiency):
   - забезпечення необхідного рівня продуктивності відповідно до виділених ресурсів, часу та інших умов.
5. Зручність супроводу (Maintainability):
   - легкість аналізу, тестування та змін для виправлення дефектів, реалізації нових вимог та полегшення подальшого обслуговування, а також адаптація до змін навколишнього середовища.
6. Портативність (Portability):
   - легкість перенесення програмного забезпечення з одного оточення в інше.

Отже, концепція якості програми представлена різноманітними характеристиками, які можна структурувати в модель якості для адекватного вираження.

Спочатку широко відомою моделлю якості програмного забезпечення була та, що була запропонована Макколом та іншими у 1977 році. Ця модель класифікує характеристики якості на три основні групи [19]:

1. Фактори (Factors):
   - описують програмне забезпечення з точки зору користувача та заданих вимог;
   - складаються з 11 факторів, які групуються відповідно до виду роботи з програмним забезпеченням. Структура цих факторів представлена у вигляді трикутника Маккола.
2. Критерії (Criteria):
   - описують програмне забезпечення з точки зору розробника та визначаються як цілі;
   - представляють собою числові рівні факторів, які встановлюються як цілі під час розробки.
3. Метрики (Metrics):
   - використовуються для кількісного опису та вимірювання якості програмного забезпечення;
   - введені Макколом для полегшення вимірювання та оцінки факторів якості, оскільки їх об'єктивна оцінка є важливим завданням. Оцінки за шкалою знаходяться в діапазоні від 0 до 10.

Метрики якості програмного забезпечення включають:

1. Зручність перевірки на відповідність стандартам (Auditability).

Оцінює, наскільки легко можна перевірити відповідність програми стандартам.

2. Точність управління й обчислень (Accuracy).

Визначає ступінь точності управління та обчислень в програмі.

3. Ступінь стандартності інтерфейсів (Communication Commonality).
   Визначає ступінь стандартності інтерфейсів у програмі.
4. Функціональна повнота (Completeness).
   Оцінює, наскільки програма включає в себе всі необхідні функції.
5. Однорідність використовуваних правил проектування й документації (Consistency):

Визначає, наскільки однорідні та спрямовані на правила проектування та документації в програмі [20].

6. Ступінь стандарності форматів даних (Data Commonality).
   Оцінює, наскільки стандартизовані формати даних у програмі.
7. Стійкість до помилок (Error Tolerance).

Визначає, наскільки програма може ефективно працювати попри наявність помилок.

8. Ефективність роботи (Execution Efficiency).
   Визначає ефективність виконання програми за виділеними ресурсами.
9. Розширюваність (Expandability).
   Оцінює легкість розширення функціоналу програми.
10. Широта сфери потенційного використання (Generality).
    Визначає, наскільки широкий спектр використання програми.
11. Незалежність від апаратної платформи (Hardware Independence).

Оцінює, наскільки програма незалежна від конкретної апаратної платформи.

12. Повнота протоколювання помилок й інших подій (Instrumentation).
    Визначає, наскільки повно програма реєструє помилки та інші події.
13. Модульність (Modularity).
    Визначає ступінь модульності програми.
14. Зручність роботи (Operability).
    Оцінює, наскільки легко користувач може взаємодіяти з програмою.
15. Захищеність (Security).
    Визначає рівень захищеності програми від несанкціонованого доступу.
16. Самодокументованість (Self-documentation).

Оцінює, наскільки програма здатна документувати свій власний код та функціонал.

17. Простота роботи (Simplicity).
    Визначає, наскільки просто користувач може взаємодіяти з програмою.
18. Незалежність від програмної платформи (Software System Independence).

Визначає, наскільки програма незалежна від конкретної програмної платформи.

19. Можливість порівняння проекту з вимогами (Traceability).

Оцінює, наскільки легко можна порівняти реалізований проект з вимогами.

20. Зручність навчання (Training) [20]:

Визначає легкість, з якою користувач може оволодіти використанням програми через навчання.

Кожна метрика впливає на оцінку ряду факторів якості, представлену числовою характеристикою. Фактор може бути представлений як лінійна комбінація значень різних метрик, що впливають на нього. Коефіцієнти цієї комбінації визначаються залежно від конкретної організації, розробницької команди та характеристик конкретного програмного забезпечення.

Рисунок 3.1 – Трикутник Маккола

Поняття якості програмного забезпечення може бути розчленоване на внутрішню якість, пов'язану із характеристиками самого програмного забезпечення, відокремленою від його поведінки. Також розрізняється зовнішня якість, яка визначається з точки зору поведінки програми та враження користувача в різних контекстах роботи з програмним забезпеченням.

Для оцінки цих аспектів якості використовуються метрики. Крім того, важливо враховувати якість технологічних процесів розробки програмного забезпечення для створення високоякісного продукту.

В сфері управління якістю видатні постаті, такі як Джозеф Джуран, Філіп Кросбі і Едвард Демінг, відомі своїми визначеннями якості. Кожен з них має своє розуміння якості, проте спільною є їхня увага до сприйняття клієнтом якості продукту [20].

Наприклад, Джозеф Джуран в 1988 році ввів визначення якості як «придатність до використання» або «фітнес для використання» – якість, що

відповідає вимогам клієнта. Він також підкреслює важливість створення продукції, вільної від дефектів, оскільки дефекти призводять до неприємностей для клієнтів та їх незадоволення. Визначення Джурана відображає його фокус на задоволенні очікувань клієнта.

Філіп Кросбі визначає якість як відповідність вимогам («Вимоги сумісності», 1979), підкреслюючи, що для нього якість або присутня, або відсутня. Він відкидає ідею різних рівнів якості, підтримуючи просту бінарну оцінку.

Загалом, можна виділити два види якості в контексті Кросбі:

1. Зовнішня якість:
   - орієнтована на клієнта;
   - визначається зручністю використання, відсутністю помилок, високою продуктивністю тощо.
2. Внутрішня якість:
   - орієнтована на розробників програмного продукту;
   - визначається відповідністю вимогам, зручною архітектурою, простотою внесення змін тощо.

Такий підхід до визначення якості відображає Кросбів погляд на якість як на концепцію, яка може бути однозначно визначена та виміряна, а також як на явище без різних рівнів.

3.2 Життєвий цикл програмного забезпечення

Тестування не є самостійним процесом, а становить невід’ємну частину моделі життєвого циклу програмного забезпечення (Software Development Life Cycle, SDLC). Таким чином, вибір засобів і методик тестування напряму визначається обраною моделлю розробки.

Життєвий цикл програмного забезпечення представляє собою умовну схему, що включає різні етапи, які відображають стадії процесу створення програмного забезпечення.

На кожному етапі циклу виконуються різні дії. Життєвий цикл програмного забезпечення складається з послідовних стадій, важливіші з яких включають.

1. Визначення потреб.
2. Аналіз вимог і формулювання концепції.
3. Розробка.
4. Виробництво.
5. Впровадження/продаж.
6. Експлуатація.
7. Супровід і підтримка.
8. Вилучення з експлуатації.

Кожна з цих стадій подальш деталізується на більш дрібні етапи. Моделі життєвого циклу програмного забезпечення детально описують взаємозв'язки між цими стадіями [20].

Існує кілька відомих типів моделей життєвого циклу програмного забезпечення, серед яких найбільш поширені – послідовні та ітераційні. Практично ці моделі можуть поєднуватися, утворюючи змішані варіанти. Цикл розробки пропонує шаблон, використання якого сприяє простоті проектування, створенню і випуску високоякісного програмного продукту. Це методологія, що визначає необхідні процеси та ресурси для успішного завершення проекту.

Хоча втілення принципів побудови моделі життєвого циклу може суттєво відрізнятися між різними компаніями, існують стандарти, такі як ISO/IEC 12207, що визначають прийняті практики в галузі розробки і підтримки програмного забезпечення.

Основна мета використання моделей життєвого циклу – створити ефективний, економічно вигідний і якісний програмний продукт.

На відміну від моделей з приростом, еволюційні підходи застосовуються там, де неможливо визначити всі вимоги одразу або коли відомо, що вони можуть змінитися. Процес розробки проекту за такими моделями також включає ітерації, проте кожна ітерація охоплює всі стадії розробки - від аналізу вибраного набору вимог до випуску версії. Кожна ітерація включає прототипування вимог і проекту.

До найбільш відомих еволюційних моделей належать спіральна модель і модель еволюційного прототипування.

Спіральна модель життєвого циклу, розроблена Боемом, включає в себе переваги каскадної моделі, додавши аналіз ризиків, їх управління, а також процеси підтримки та менеджменту. Вона також передбачає розробку програмного продукту за допомогою методу прототипування чи швидкої розробки додатків з використанням мов програмування та засобів четвертого покоління та інших. Базова концепція моделі полягає в тому, що кожен цикл представляє собою набір операцій, і для кожної операції визначається відповідна кількість стадій, як у моделі каскадного процесу. Кожна складова частина продукту та рівень абстракції розглядаються від формулювання потреб до кодування окремих програм [20].

3.3 Типи тестування мобільних додатків

Створення мобільного додатка під Android – завдання, що несе в собі свої власні виклики та особливості. Є декілька ключових аспектів, які варто мати на увазі.

По-перше, варто звернути увагу на велику різноманітність пристроїв, що працюють під управлінням Android. Ця різноманітність дійсно вражає. Для користувачів це надає можливість вибору смартфону, що відповідає їхнім потребам і технічним вимогам. Але для розробників додатків це становить величезний виклик як з погляду апаратної, так і програмної сумісності.

Особливу увагу слід звернути на розміри екрану та їхню роздільну здатність. Наприклад, коли ви хочете оптимально відобразити зображення на весь екран iOS, ви зазвичай підготовляєте кілька версій зображень для різних моделей iPhone, такі як iPhone 6 і новіші, iPhone 6 Plus і новіші, iPhone X і iPhone X Max. У випадку з Android, різні пристрої мають різні роздільні здатності, співвідношення сторін і щільність пікселів.

По-друге, іншою важливою складністю є широкий спектр версій операційної системи Android, які встановлені на пристроях користувачів. Це призводить до численних проблем, коли розробникам доводиться створювати мобільні додатки «з нуля» для Android:

A. Під час розробки додатка слід уважно розглядати відображення інтерфейсу на різних версіях операційної системи Android і різних оболонках. Системні елементи управління можуть значно відрізнятися між різними версіями Android і оболонками навіть в межах однієї версії Android.
B. Різні версії операційної системи мають свою особливу логіку роботи в ряді аспектів. Наприклад, до версії 6.0 додатки могли не запитувати окремо кожен дозвіл (наприклад, доступ до камери, мікрофону і т. д.), а вказували їх у списку в Google Play, припускаючи, що користувачі знайомі з цими дозволами перед завантаженням додатка. Починаючи з версії 6.0, кожен дозвіл повинен запитуватися окремо в процесі роботи додатка. Якщо не враховувати обидві варіанти логіки при розробці мобільного додатка для Android, це може призвести до некоректної роботи додатка на версіях до 6.0 або пізніших версіях.
C. Програмні методи та бібліотеки постійно змінюються, деякі з них вважаються застарілими і потребують заміни на більш сучасні альтернативи. Тому розробникам завжди доводиться вибирати між підтримкою найновіших функцій операційної системи і максимальною сумісністю з більшою кількістю користувачів.
D. Останні версії операційної системи Android підтримують багатозадачність на робочому столі, дозволяючи користувачам одночасно відображати кілька додатків та вибирати розмір робочої області за власними потребами. Цей аспект також має бути врахований під час створення додатків.

Тестування мобільних додатків можна поділити на такі типи:

1. Функціональне тестування.

Функціональне тестування перевіряє, чи працюють функції згідно з вимогами. Наприклад, воно тестує взаємодію користувача з додатком, такі як запуск програми, вхід у систему, відтворення пісні, перевірка балансу облікового запису та інші прості користувацькі дії. Цей процес включає в себе різні аспекти, такі як інтерфейс програми, база даних, мережа і багато інших. Для досягнення найкращих результатів потрібно збалансувати різні типи функціональних тестів.

2. Регресійне тестування.

Регресійне тестування перевіряє, чи нові оновлення, виправлення або зміни не спричиняють нових помилок або регресій, як у функціональних, так і нефункціональних аспектах мобільного додатка. Воно забезпечує впевненість, що зміни, внесені в процесі розробки, не порушують роботу програми. Наприклад, компанії, які надають послуги програмного забезпечення як сервіс, регулярно оновлюють свої додатки, і регресійне тестування гарантує, що ці оновлення не завдають шкоди основному продукту.

3. Тестування продуктивності.

Тестування продуктивності мобільного додатка визначає, як він реагує на навантаження. Цей процес оцінює швидкість, стабільність і масштабованість програми як на стороні клієнта, так і на стороні сервера. На стороні сервера воно досліджує зміни часу відповіді, потік ресурсоємних запитів, затримки доставки повідомлень та інші параметри. На стороні клієнта воно перевіряє поведінку додатку на різних платформах, витрату пам'яті та обчислювальних ресурсів, швидкість завантаження та проблеми з батареєю.

4. Тестування безпеки.

Безпека є критично важливою для мобільних додатків, оскільки вона забезпечує захист даних користувачів. Тестування безпеки вимагає знань у різних областях, таких як комунікація клієнт-сервер, архітектура програмного забезпечення та архітектура системи. Цей процес включає в себе методи, такі як тестування на проникнення, фаззинг, сканування та аудит програмного забезпечення для виявлення потенційних вразливостей.

5. Юзабіліті-тестування.

Під час юзабіліті-тестування реальні користувачі оцінюють, наскільки легко використовувати мобільний додаток. Тут увага приділяється спрощенню використання, легкому підключенню та задоволенню користувача. Тестування включає в себе завдання, які користувачам необхідно виконати, щоб визначити, наскільки добре додаток задовольняє їхні потреби.

6. Тестування сумісності.

З огляду на різноманітність мобільних пристроїв і платформ, тестування сумісності стає необхідним. Воно перевіряє, як додаток працює на різних мобільних пристроях та браузерах, щоб впевнитися в його належному функціонуванні.

3.4 Проблеми тестування мобільних додатків

Мобільні додатки повинні працювати безперебійно на різних пристроях та версіях операційних систем. Інженери відділу забезпечення якості (QA) мають негайно реагувати на будь-які зміни в апаратному та програмному забезпеченні, які можуть призвести до проблем для користувачів. Командам потрібно враховувати всі аспекти, від налаштувань

мережевого оператора до рівня заряду батареї. Фрагментація стає особливою проблемою, коли деякі користувачі продовжують використовувати старі версії операційних систем або пристроїв, навіть коли вже є нові.

Тестування кожної можливої комбінації пристрою, ОС та мережевих налаштувань створює велику кількість тестових випадків. Це вимагає від груп розробників постійного вдосконалення та підтримки різноманітних мобільних пристроїв. Ці аспекти стають серйозною перешкодою для розробників мобільних додатків.

Більшість мобільних додатків використовують різні типи мережевого підключення, включаючи передачу даних (Edge, UMTS, 3G, 4G) та Wi-Fi. Пересуваючись, користувачі можуть змінювати тип підключення або втрачати його зовсім. Деякі оператори зв'язку фільтрують доступ до Інтернету, що може призвести до ситуацій, коли пристрій підключений, але фактично немає доступу до певних послуг (наприклад, обмін повідомленнями чи дзвінки через додатки). Реальне середовище неможливо повністю відтворити, і проблеми можуть виникнути навіть при наявності відповідних API для підключення на мобільних платформах. Тому розробникам QA важливо тестувати використання смуги пропускання, оскільки не всі оператори пропонують необмежену передачу даних.

Багато мобільних додатків вимагають сторонніх інтеграцій для аналітики, звітів про збої та SMS-сервісів. Щоб забезпечити безперешкодну взаємодію з користувачем та правильну функціональність, ці додатки значно віддають залежать від сторонніх інтеграцій та апаратних компонентів. Це змушує команди тестування перемикатися між різними об'єктами або пристроями під час тестування. Складні сценарії використання підвищують обсяг роботи, потрібний для належного тестування програми.

Мобільні додатки з ігровим аспектом або потоковим відео можуть швидко розряджати акумулятор. Протягом дня користувачі запускають безліч додатків, а кілька процесів може працювати в фоновому режимі, споживаючи ресурси і енергію батареї. Це призводить до виснаження акумулятора. Спеціалізована лабораторія для мобільних пристроїв використовується для вивчення впливу нових кодових змін, впливу на споживання пам'яті телефону, швидкість роботи та витрату заряду батареї [20].

3.5 Загальна функціональність

Функціональне тестування є важливою складовою процесу визначення відповідності поведінки системи вихідним функціоналу. Його мета полягає в підтвердженні того, що реалізація системи відповідає визначеним функціональним вимогам та повністю готова до ефективної роботи.

Функціональне тестування є етапом тестування програмного забезпечення, спрямованим на перевірку того, чи відповідає

функціональність системи початково визначеним вимогам. Основною метою функціонального тестування є підтвердження, що система працює відповідно до зазначених функціональних вимог та є готовою до повноцінної роботи.

А. Доступ до камери.

Один із важливих аспектів функціонального тестування – перевірка функціональності доступу до камери в додатку. Це включає в себе перевірку відповідності додатка початковим вимогам, пов'язаним із взаємодією з камерою пристрою.

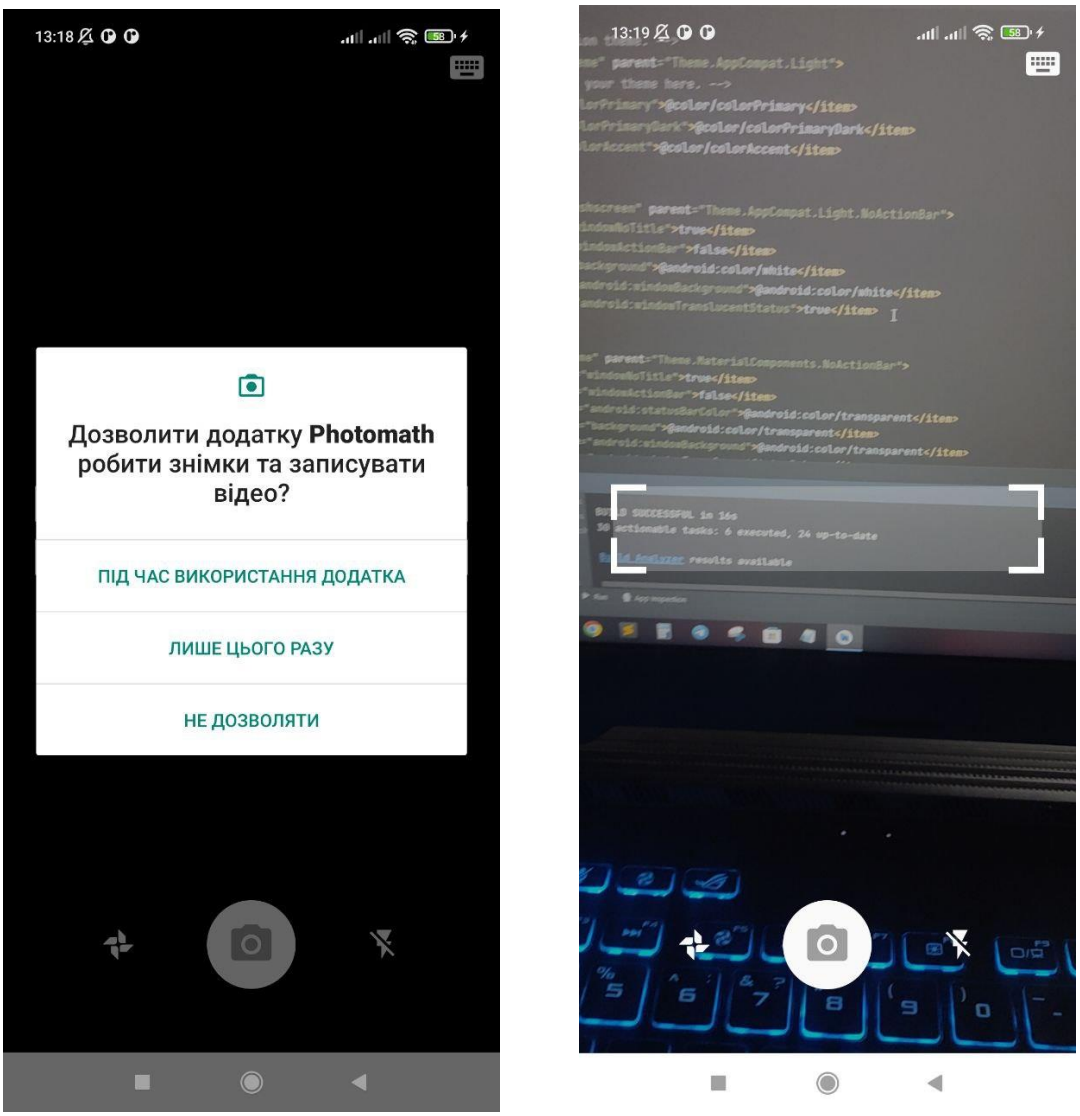

Рисунок 3.2 – Доступ до камери

Проведене тестування підтвердило, що додаток працює відповідно до встановлених вимог. У випадку, якщо користувач не дозволяє додатку використовувати камеру, з'являється відповідне діалогове вікно, і програма припиняє свою роботу. Користувачу надається можливість повторно підтвердити дозвіл для продовження використання додатку. Результати тестування функціональності доступу до камери свідчать про відповідність

додатка встановленим стандартам та готовність до ефективної взаємодії з користувачем.

B. Використання пам’яті та Інтернету.

Використання ресурсів, таких як пам'ять і Інтернет, є ключовим аспектом при розробці та використанні додатків. Було проведено огляд обсягу пам'яті, який займає додаток, і його взаємодії з мережею Інтернет.

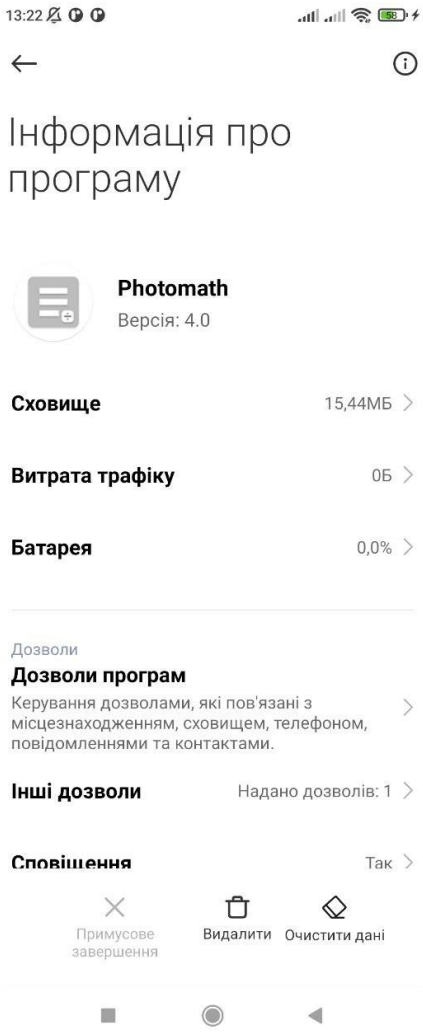

Рисунок 3.3 – Використання пам’яті та Інтернету

Одним із важливих параметрів ефективності додатку є обсяг зайнятої пам'яті. Зазначено, що даний додаток займає лише 15 МБ пам'яті, що на 45 МБ менше, ніж у його оригіналу. Менший обсяг пам'яті може призвести до зменшення конфліктів з іншими додатками та покращити швидкість виконання функцій. Затримки через низький обсяг доступної пам'яті можуть бути вирішені завдяки ефективній роботі додатку.

Забезпечення ефективного використання пам'яті та оптимізація взаємодії з Інтернетом є ключовими компонентами успіху додатку. Менший обсяг пам'яті може призвести до поліпшення продуктивності, а оптимізована взаємодія з Інтернетом забезпечить ефективну передачу даних.

C. Тестування основних функцій.

У сучасному світі розробка мобільних додатків вимагає не тільки високої продуктивності, але й забезпечення якісного тестування основних функцій. У цьому контексті проводження тестів на розпізнавання друкованого тексту, використовуючи мобільний Photomath, є ключовим етапом для впевненості у правильній роботі додатка.

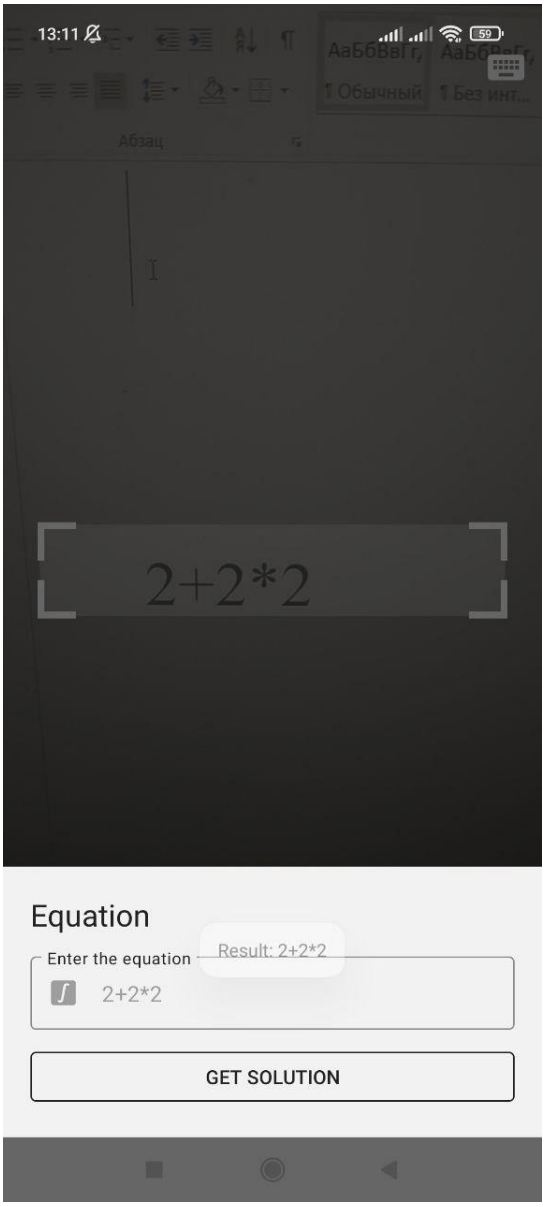

Рисунок 3.4 – Сканування з екрану монітора

ML Kit, який використовується в додатку, є потужним інструментарієм для обробки машинного навчання в мобільних додатках. Вибір цього SDK для розпізнавання друкованого тексту має свої переваги, але обмеження відображення лише друкованого тексту може вплинути на функціональність додатка.

Оскільки основна функція додатка пов'язана з розпізнаванням друкованого тексту, проведення тестів для перевірки цієї функціональності стає обов'язковим. Застосування тестів на реальних даних, таких як вирази,

що скануються з екрана монітора, є важливим кроком для визначення ефективності розпізнавання та коректності виведення результатів.

У рамках важливих етапів тестування функціональності додатка було проведено детальну перевірку отримання рішень і роботи звичайного калькулятора. Цей процес спрямований на визначення ефективності та точності роботи калькулятора в різних сценаріях використання.

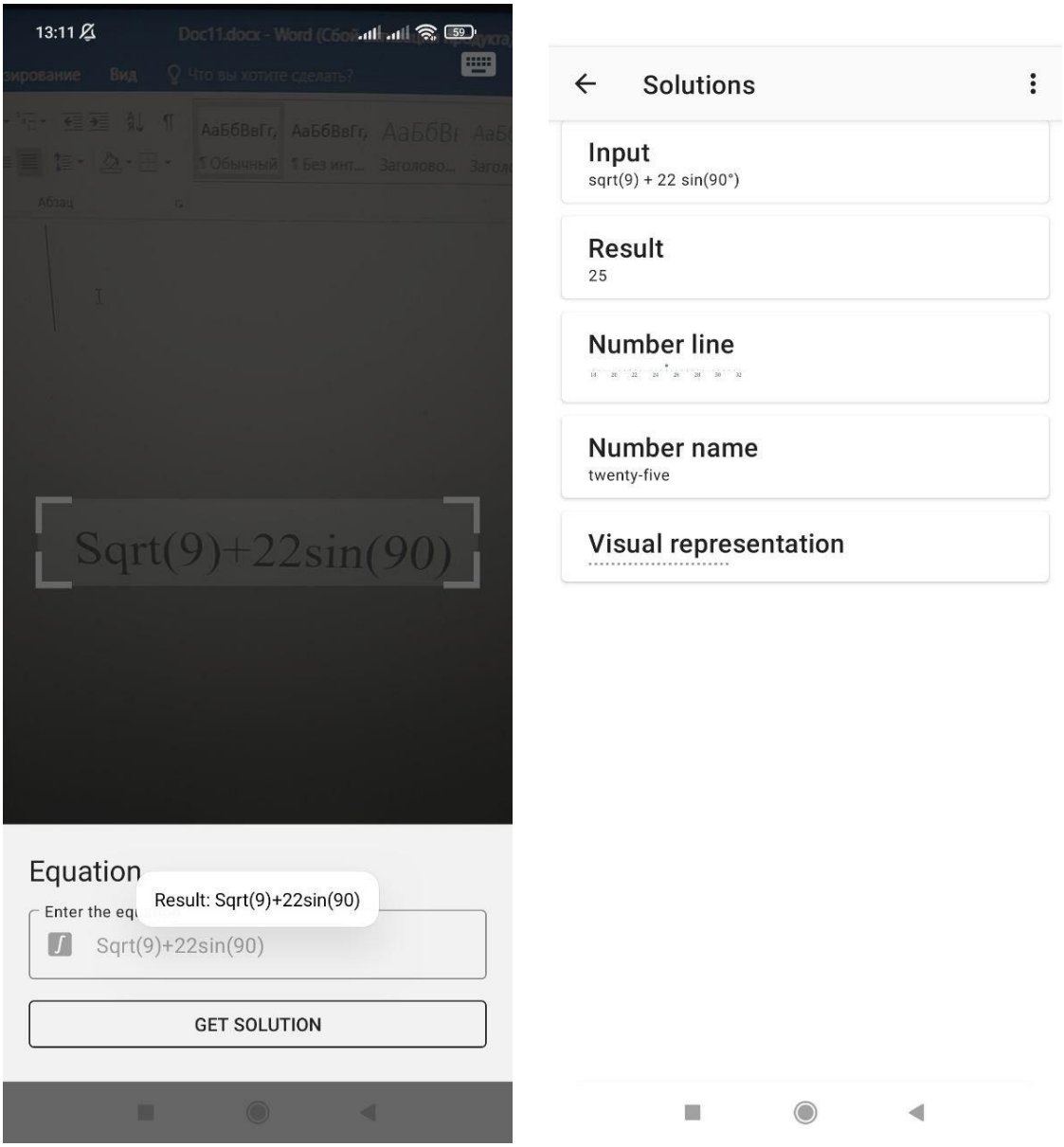

Рисунок 3.5 – Перевірка отримання рішення простого виразу

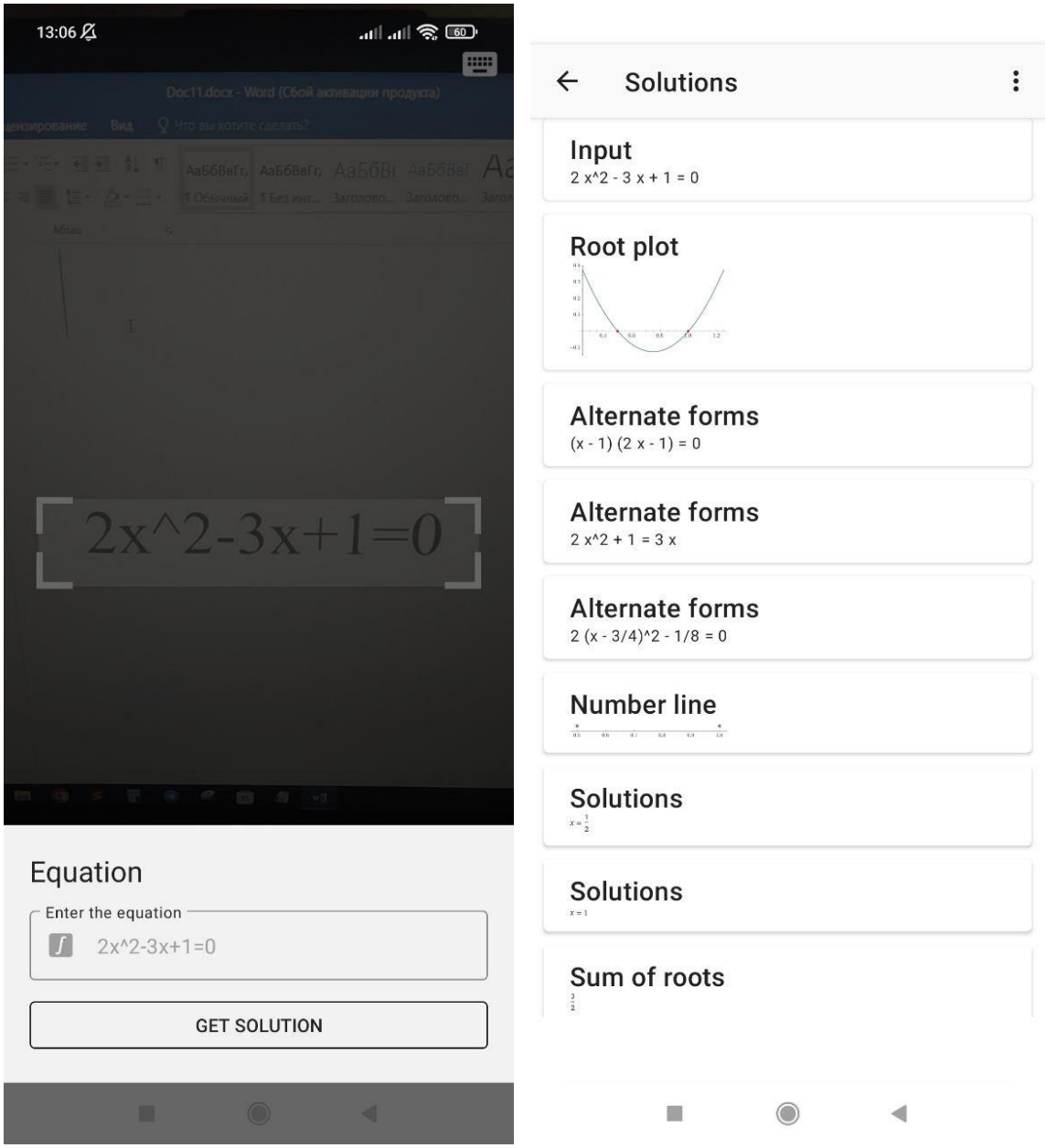

Рисунок 3.6 – Перевірка отримання рішення складного виразу

У процесі розробки та тестування додатків виникає необхідність в перевірці роботи з медіа-контентом, зокрема – фотографіями. Важливим етапом стало тестування отримання та форматування фотографій з файлового сховища користувача.

Додаток, який взаємодіє з фотографіями, повинен ефективно взаємодіяти з файловим сховищем користувача. Тестування цього аспекту включало в себе спробу отримати фотографії з різних частин файлового сховища, визначити швидкість та ефективність операцій отримання.

Тестування процесу форматування фотографій включало в себе спостереження за тим, як додаток взаємодіє з фотографіями різного розміру та якості. Застосування різних операцій форматування, таких як зміна розміру та налаштування якості, має забезпечити консистентність та високу якість відображення фотографій.

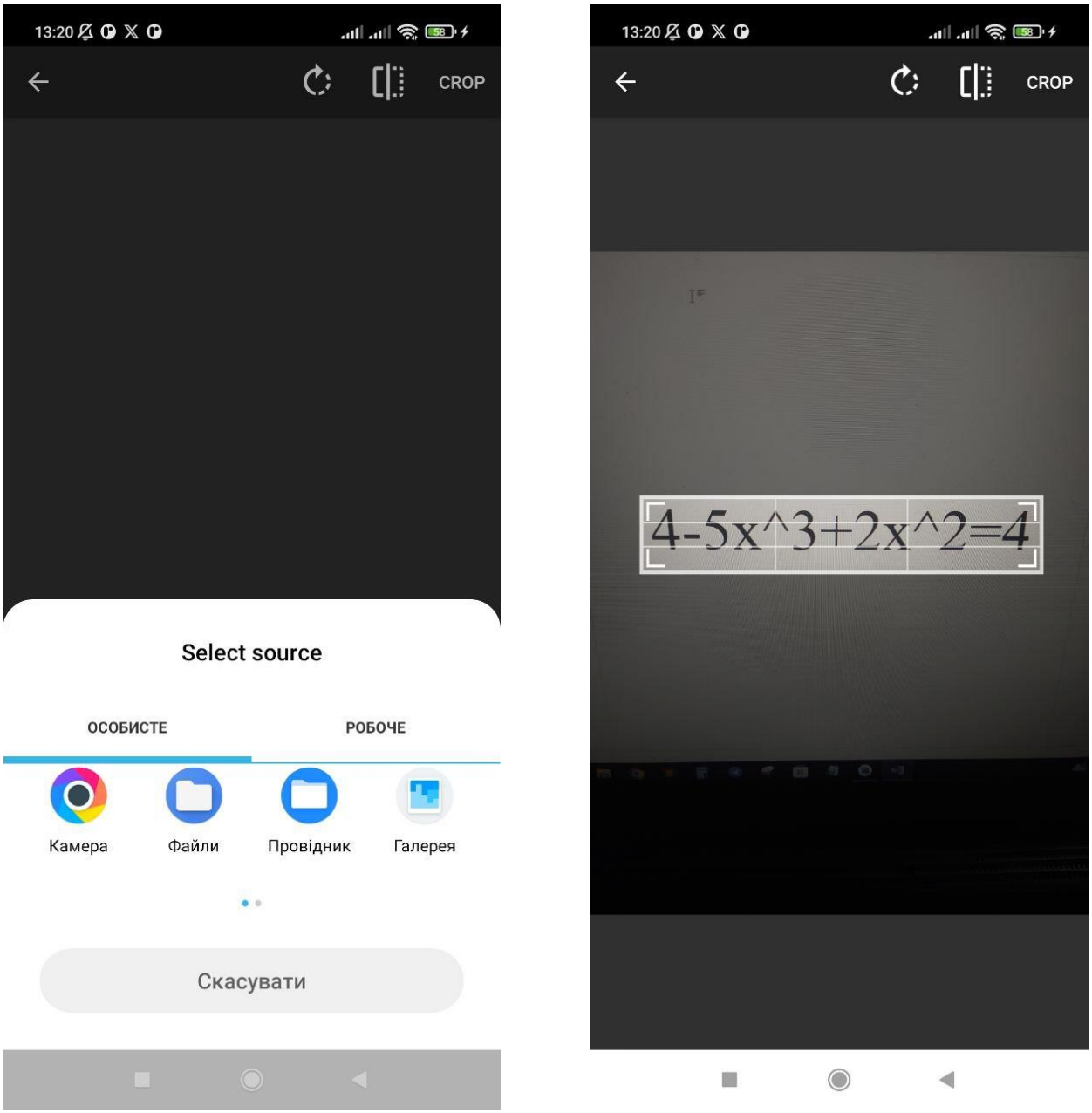

Рисунок 3.7 – Отримання та форматування фотографій з файлового сховища користувача

3.6 Тестування сумісності

Один з ключових етапів тестування додатка – це перевірка його сумісності з різними версіями Android-платформи. Вказуючи мінімальний рівень API для роботи додатка, розробники визначають мінімальну версію операційної системи, на якій можна очікувати стабільну роботу.

У рамках цього етапу тестування було проведено на різних смартфонах – Meizu M5s (з версією API 23) та Huawei P Smart Z (з версією API 29). Важливо відзначити, що вибір цих пристроїв був стратегічним, охоплюючи як старіші, так і новіші версії Android.

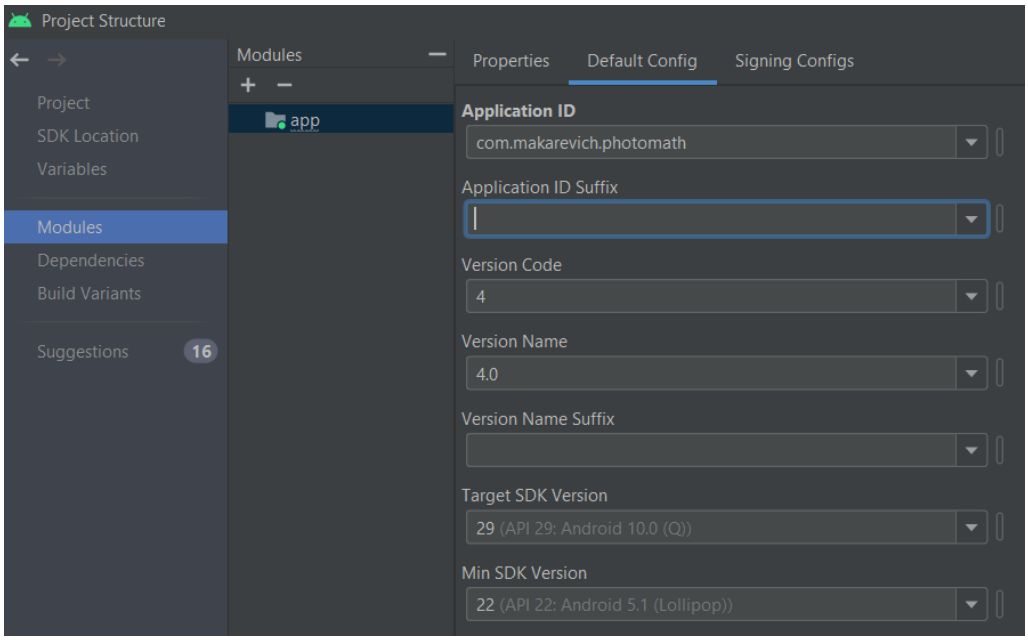

Рисунок 3.8 – SDK Versions

Під час тестування було перевірено, чи виконуються всі функції додатка коректно на обраних пристроях. Важливо враховувати, що різні версії API можуть викликати різні виклики до системи та мають різні обмеження. Отже, перевірка коректності функцій дозволяє забезпечити, що додаток працює стабільно на всіх підтримуваних версіях Android.

Однією з особливостей сумісності додатка є його адаптація до різних роздільних здатностей екранів. У тестуванні було враховано різні роздільні здатності обраних пристроїв, щоб переконатися, що інтерфейс додатка виглядає коректно та зручно на екранах різного розміру та роздільної здатності.

Висновки за розділом

В третьому розділі кваліфікаційної роботи було проведено тестування мобільного додатку.

ВИСНОВОК

В ході виконання кваліфікаційної роботи було проведено докладний аналіз програмних засобів та технологій для реалізації Photomath на платформі Android. Робота розглянула основні аспекти розробки Android-додатків, включаючи структуру програми, віртуальні машини, управління ресурсами та життєвий цикл візуальних компонентів.

Аналіз існуючих рішень, таких як Photomath, Mathway: Scan & Solve Problems і Microsoft Math Solver, визначив актуальні концепції та підходи до реалізації подібних додатків. Постановка завдання та вибір методів реалізації були проведені з урахуванням найкращих практик та новітніх технологій в галузі мобільної розробки.

Результатом роботи стало успішне створення мобільного додатка Photomath для платформи Android, який включає в себе реалізацію основного функціоналу, такий як налаштування камери, розпізнавання тексту та створення калькулятора. Розгорнуте тестування додатка охопило якість програмного забезпечення, життєвий цикл та сумісність з різними пристроями.

Завершуючи дослідження, можна визначити, що отримані результати мають важливе практичне значення для розробників мобільних додатків, особливо в галузі освітніх технологій. Реалізація Photomath на платформі Android дозволяє користувачам отримати доступ до високоякісної математичної допомоги в будь-який момент і місце. Розроблені методи та моделі можуть бути використані в подальших дослідженнях у галузі розвитку освітніх додатків та розширення їхнього функціоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цехнер М. Программирование игр под Android. СПб. : Питер, 2005. 688 с.
2. Фелкер Д. Android: разработка приложений для чайников. Москва: ООО “И.Д.Вильямс”, 2012. 336 с.
3. Дейтел П., Дейтел Х., Уолд А. Андроид для разработчиков. СПб. : Питер, 2016. 512 с.
4. Медникс З., Дорнил Л., Мик Б., Накамура М. Программирование под Android. СПб. : Питер, 2013. 560 с.
5. МакГрат М. Программирование на Java для начинающих. Москва : “Э”, 2016. 192 с.
6. Criado M. Design and construction of a learning game on Android / Marc Criado. – Barcelona: BarcelonaTech, 2015. – 112 с.
7. Zachner M. Beginning Android Games / M. Zachner, R. Green. – 706 с.
8. Bassey A. Online Gaming: A Successful Life? The Influence of Gaming on Development / Angela Bassey. – Kongsberg, 2020. – 86 с.
9. Morgan G. Challenges of Online Game Development: A Review / Graham Morgan. – Newcastle Upon Tyne, 2009. – 23 с.
10. Gerber A. Learn Android Studio / A. Gerber, C. Craig., 2010. – 470 с.
11. Kumar S. An Introduction to Client/Server computing / S. Kumar, S. Chandra. – New Delhi, 2009. – 213 с.
12. Rogers R. Android Application Development / R. Rogers, J. Lombardo. – California, 2009. – 336 с.
13. Зуев В. А. Разработка мобильных приложений: методические указания к выполнению практических занятий и лабораторных работ / В. А. Зуев. – Новочеркасск, 2019. – 68 с.
14. Крепич С. Я. Якість програмного забезпечення та тестування / С. Я. Крепич, І. Я. Співак. – Тернопіль, 2020. – 479 с.
15. Пацей Н. В. Разработка мобильных приложений / Н. В. Пацей. – Минск, 2020. – 265 с.
16. Габедава О. В. Серверные технологии / О. В. Габедава, С. М. Почовян. – Тбилиси, 2010. – 130 с.
17. Федотенко М. А. Разработка мобильных приложений / М. А. Федотенко. – Москва, 2020. – 336 с.
18. Востокин С. В. Архитектура современных распределённых систем / С. В. Востокин. – Самара, 2013. – 91 с.
19. Марсикано К. Android. Программирование для профессионалов / К. Марсикано, Б. Филлипс. – Санкт-Петербург, 2021. – 704 с.

20. Авраменко А. С. Тестування програмного забезпечення. Навчальний посібник / А. С. Авраменко. – Черкаси, 2017. – 284 с.