МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ANDROID ЗАСТОСУНКУ ЗАСОБАМИ KOTLIN ТА FIREBASE ML KIT»**

Виконав: студент 2 курсу, групи 8.1212-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
С.Ю. Єгоров
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент
Лісняк А.О.
____
(підпис)
“ ______ ” __________ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Єгорову Сергію Юрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка Android застосунку засобами Kotlin та Firebase ML Kit

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 643-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   1. Постановка задачі.
   2. Основні теоретичні відомості
   3. Розробка Android застосунку засобами Kotlin та Firebase ML Kit.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень) ________
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	18.05.2023	
2.	Збір вихідних даних.	09.06.2023	
3.	Обробка методичних та теоретичних	30.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	28.08.2023	
5.	Розробка третього розділу.	23.10.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи магістра.	15.12.2023	

Студент ____________ С.Ю. Єгоров
(підпис) (ініціали та прізвище)

Керівник роботи ____________ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер ____________ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка Android застосунку засобами Kotlin та Firebase ML Kit»: 62 с., 42 рис., 4 табл., 22 джерела.

АКТИВІТІ (АКТИВНІСТЬ), ЗАСТОСУНОК, МАШИННЕ НАВЧАННЯ, ОБЛИЧЧЯ, ANDROID, AUTHENTICATION, FACEDETECTOR, FIREBASE, KOTLIN, ML KIT.

Об’єкт дослідження – розробка Android застосунку з використанням мови програмування Kotlin та інтеграцію сервісів Firebase ML Kit.

Мета роботи: створення безпечного та функціонального застосунку, здатного використовувати технології машинного навчання для визначення обличчя та забезпечення автентифікації через Firebase.

Метод дослідження: аналіз, проєктування, тестування.

У кваліфікаційній роботі досліджувалися наступні проблеми, спрямовані на створення сучасного та функціонального Android застосунку, використовуючи мову програмування Kotlin та інтегруючи сервіси Firebase ML Kit. Однією з ключових проблем було вирішення завдань, пов'язаних з ефективністю, безпекою та використанням передових технологій машинного навчання для покращення функціональності застосунку.

У ході дослідження було використано ряд методів, включаючи аналіз існуючих застосунків, проєктування та розробку, тестування та оцінку функціональності застосунку. Особлива увага була приділена використанню мови Kotlin як сучасного інструменту для розробки Android додатків.

Результатом роботи є створений застосунок, який не лише відповідає високим стандартам, але й відкриває нові перспективи для застосування передових технологій в області мобільного програмування та машинного навчання.

SUMMARY

Master’s qualifying paper «Development of the Android Application Using Kotlin and Firebase ML Kit»: 62 pages, 42 figures, 4 tables, 22 references.

ACTIVITY, APPLICATION, MACHINE LEARNING, FACE, ANDROID, AUTHENTICATION, FACEDETECTOR, FIREBASE, KOTLIN, ML KIT.

The object of the study is the development of an Android application using the Kotlin programming language and the integration of Firebase ML Kit services.

The aim of the study is to create a secure and functional application capable of utilizing machine learning technologies for facial recognition and ensuring authentication through Firebase.

The methods of research are analysis, design, and testing.

The qualification paper addressed challenges focused on creating a modern and functional Android application, utilizing the Kotlin programming language and integrating Firebase ML Kit services. One of the key challenges was addressing tasks related to efficiency, security, and the use of advanced machine learning technologies to enhance the application's functionality.

During the research, a range of methods was employed, including the analysis of existing applications, design and development, testing, and evaluation of the application's functionality. Special attention was given to the use of Kotlin as a modern tool for developing Android applications.

The outcome of the work is the developed application, which not only meets high standards but also opens up new perspectives for the application of advanced technologies in the field of mobile programming and machine learning.

ЗМІСТ

Завдання на кваліфікаційну роботу……..……………………………….. 2

Реферат……………………………………………………………………… 4

Summary...……………………………………………………………………… 5

Вступ…………………………………………………………………………… 8

1 Огляд платформ та мов програмування для розробки мобільних додатків…………..………..……………………..…………………..…… 9

1.1 Платформа Android..…...………………………………….………... 9

1.1.1 Мова програмування Java для Android.…………..………… 9

1.1.2 Мова програмування Kotlin …………..…..……..……... 11

1.1.2.1 Сфера застосування мови Kotlin…........…………….. 12

1.1.2.2 Переваги програмування на Kotlin……..……... 13

1.1.2.3 Перспективи розробки на Kotlin……..…………... 13

1.1.2.4 Поєднання Jetpack Compose з мовою Kotlin…….. 14

1.2 Платформа Flutter та мова програмування Dart…………………... 16

1.3 Платформа Xamarin…………………………………………… 17

1.4 Платформа React Native (RN)……………………………… 18

1.5 Платформа Firebase……………………………………………… 19

2 Сервіси та технології використані в проєкті……………………………. 23

2.1 Сервіс Firebase Authentiation……………………………………….. 23

2.2 Роль та важливість машинного навчання в розробці мобільних застосунків……………..…………………………………………... 25

2.3 ML Kit FaceDetector…………………………………………………… 26

2.3.1 Концепції розпізнавання обличчя………………………… 26

2.3.2 Орієнтація, контури, класифікація та мінімальний розмір обличчя………………………………………………………… 27

2.3.3 Можливості та технічні аспекти розпізнавання обличчя... 31

2.3.4 Розробка додатків для розпізнавання обличчя…………... 32

2.3.5 Порівняння з іншими сервісами………………………….... 33
3 Розробка застосунку засобами Kotlin та Firebase ML Kit………………… 37
3.1 Створення проєкту та його налаштування……………………….... 37
3.2 Інтеграція сервісу Firebase Authentication в проєкт………………. 38
3.2.1 Розробка коду для інтеграції сервісу Firebase Authentication……………….…….………………………………... 38
3.2.2 Тестування розробленого коду……………………………. 41
3.3 Основний функціонал проєкту…………………………………….. 43
3.3.1 Розробка коду для реалізації тесту……………………….... 43
3.3.2 Тестування розробленого коду……………………………. 47
3.4 Інтеграція сервісу ML Kit FaceDetector а проєкт…………………. 48
3.4.1 Розробка коду для реалізації сервісу від ML Kit FaceDetector ……………………………………………………… 48
3.4.2 Тестування розробленого коду……………………………. 56
Висновки……………………………………………………………………….. 59
Перелік посилань………………………………………………….....…….. 60

ВСТУП

В сучасному світі, де технології стрімко розвиваються, виникає необхідність у створенні інноваційних застосунків, які не лише відповідають високим стандартам безпеки, але й використовують передові сервіси в галузі машинного навчання, такі як ML Kit. Нинішні виклики перед високотехнологічними проєктами вимагають не тільки забезпечення бездоганної безпеки, але й використання розумних інструментів для оптимізації функціональності та покращення взаємодії з користувачем.

Одним із ключових напрямків цього дослідження є створення Android застосунку на мові програмування Kotlin, який інтегрує сервіси Firebase, такі як Authentication, для забезпечення надійного та безпечного доступу, а також ML Kit з використанням сервісу FaceDetector для визначення обличчя користувачів. Це відкриває можливості для застосування штучного інтелекту в процесі тестування та взаємодії з користувачем.

Основні завдання цього дослідження полягатимуть у створенні зручного і ефективного інструменту для тестування знань або навичок користувача, забезпечуючи при цьому високий рівень безпеки та використовуючи потужний інструментарій машинного навчання. Дослідження буде спрямоване на оптимізацію процесу аутентифікації через Firebase, використання FaceDetector для аналізу параметрів обличчя, а також розробку гнучкої тестової системи, яка може бути адаптована до різних галузей та освітніх вимог.

Отже, розвиток цього застосунку має потенціал вирішити важливі завдання у сферах безпеки, машинного навчання та тестування, відповідаючи на сучасні виклики та надаючи унікальні можливості для покращення взаємодії з користувачем та розвитку освіти.

1 ОГЛЯД ПЛАТФОРМ ТА МОВ ПРОГРАМУВАННЯ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

Сьогодні мобільні додатки стали важливою частиною повсякденного життя. Існують різні платформи для розробки мобільних додатків та мови програмування. Розглянемо кожну з цих платформ.

1.1 Платформа Android

Платформа Android, яка є однією з провідних платформ для розробки мобільних додатків, продовжує еволюціонувати, а тому з'являються нові мови програмування, інструменти та технології. Основними мовами розробки для Android залишаються Java та Kotlin. Наведемо аналіз та порівняння цих двох мов в контексті розробки Android.

1.1.1 Мова програмування Java для Android

Основну сутність мови Java складають бібліотеки файлів, звані класами, кожен з яких містить невеликі фрагменти перевіреного, готового до виконання коду. Будь-які з цих класів можна вбудовувати в нову програму, і таким чином, для остаточного завершення програми залишається написати невелику частину коду. Така методика економить багато часу і є однією з основних причин широкої популярності програмування на Java. До того ж така модульна організація спрощує процес налагодження: адже знайти помилку в невеликому модулі набагато простіше, ніж в одній великій програмі. Технологія Java є одночасно і платформою, і мовою програмування. Тексти програм мови Java написані в звичайному текстовому

файлі з розширенням .java, який потім компілюється в файли з розширенням .class за допомогою компілятора javac. Після цього програма виконується інтерпретатором java за допомогою віртуальної машини Java (Java VM) [1].

Оскільки віртуальна машина Java доступна на різних платформах, одні і ті ж файли .class можна запускати, як в середовищі Windows і Linux, так і на комп'ютері Mac. Це і є основний принцип платформ мови – «написано один раз, працює скрізь». JRE забезпечує безпечну та зручну роботу додатків на Java, тому користувачі можуть не турбуватись про несанкціоноване втручання до ресурсів свого персонального комп’ютера з боку стороннього Java коду. Необхідно лише періодично оновлювати JRE. Вбудована технологія забезпечення безпеки Java включає в себе значний набір API (Application Programming Interface) механізмів та додаткових інструментів, включаючи відомі та надійні алгоритми та протоколи безпеки. Це передбачає використання криптографічних механізмів захисту інформації, інфраструктуру відкритих ключів, захищений зв'язок, автентифікацію та контроль доступу [14].

Java розширила спектр об’єктів, які можуть розповсюджуватись у кіберпросторі. Програми нової форми – аплети – завантажуються з віддаленого сервера і можуть запускатися динамічно, без участі користувача.

Аплети Java інтерпретуються, а не компілюються, і їх виконання на різних платформах значно полегшується. В цьому випадку достатньо створити для кожної платформи виконуючу Java-систему. Якщо існує така система для даної операційної системи, то будь-яка Java-програма може виконуватись в даному середовищі без додаткової компіляції на цій платформі. Java задумана для використання на різних платформах і тому реалізує в собі найбільш стандартні можливості, задля легшої адаптації під конкретне середовище.

Java це об’єктно-орієнтована мова програмування, у якій сформований ясний і прагматичний підхід до об’єктів. Java підтримує об’єктно-орієнтовану обробку виключних ситуацій подібно до C++, але на відміну від

C++ в Java обробка виключних ситуацій є обов'язковою. Тобто неможливо скомпілювати програму, яка відкриває файл, не обробивши можливі помилки.

Java розроблялася з орієнтацією на вимоги до створення інтерактивних програм, які працюють з мережею. З цією метою Java підтримує багатопотокове програмування, яке дозволяє легко розробляти програми, що виконують багато процесів одночасно.

Java є системою, яка легко розширюється за рахунок створення нових стандартних класів та бібліотек, для того щоб «одного разу написане працювало всюди, в любий час і завжди» [1].

1.1.2 Мова програмування Kotlin

Kotlin – це сучасна, багатоцільова мова програмування, яка зародилася в JetBrains і швидко набула широкої популярності серед розробників. Ця мова призначена для створення додатків на платформі Java, але вона також підтримує безліч інших цілей, включно з Android-розробкою, веброзробкою та багато іншого.

Kotlin надає розробникам великий набір конструкцій для зручнішої та ефективнішої роботи з даними та логікою програми. Включно з умовними операторами (if, when), циклами (for, while), операторами присвоювання та інше. Комбінація цих конструкцій робить код на Kotlin більш читабельним і легко підтримуваним [13].

Однією з ключових особливостей Kotlin є його сумісність із мовою програмування Java. Тобто, код на Kotlin може бути інтегрований в існуючі Java-проєкти без будь-яких проблем. Ця сумісність зумовлена не тільки схожим синтаксисом, а й тим, що Kotlin компілюється в байт-код, який може виконуватися на віртуальній машині Java (JVM). Це полегшує процес переходу на Kotlin, та дає змогу поступово впроваджувати нові елементи в

кодову базу, не переписуючи всю систему.

Kotlin застосовує статичну типізацію, що означає, що типи змінних визначаються на етапі компіляції і не можуть змінюватися під час виконання програми. Це істотно збільшує надійність коду, оскільки багато помилок і проблем можна виявити на ранніх стадіях розробки, до запуску програми. Тому, Kotlin допомагає уникати багатьох типових помилок, таких як неправильне звернення до об'єктів (NullPointerException) [5].

1.1.2.1 Сфера застосування мови Kotlin

Однією з ключових і важливих сфер застосування Kotlin є розробка мобільних додатків для платформи Android. У 2017 році Kotlin було оголошено офіційною мовою програмування для Android. Основні характеристики та переваги Kotlin:

– офіційна підтримка Google. Google офіційно підтримує Kotlin в Android Studio, надаючи плагіни та інструменти для ефективної розробки додатків під Android;
– покращена безпека. Kotlin допомагає уникнути багатьох поширених помилок, як NullPointerException, завдяки своїй системі типів і безпеки на етапі компіляції;
– скорочення обсягу коду. Kotlin дає змогу писати коротший і виразніший код порівняно з Java, що спрощує розробку та обслуговування Android-додатків;
– функціональне програмування. Можливості функціонального програмування Kotlin роблять його ідеальним вибором для розробки сучасних Android-додатків.

Kotlin також використовується у сфері серверної розробки: створення вебсервісів, API, мікросервісів та інших компонентів серверних додатків. Він добре інтегрується з такими фреймворками та бібліотеками, як Spring і Ktor.

Kotlin надає можливості для створення крос-платформних додатків. Підпроєкт мови Kotlin – Native, дає змогу розробляти desktop-додатки, які можна запустити на різних операційних системах, як Windows, macOS і Linux. Це може бути корисно для створення інструментів або додатків із невеликою клієнтською базою.

Фреймворк Ktor дає змогу створювати вебдодатки з використанням Kotlin. Це дає змогу створювати сучасні та високопродуктивні вебдодатки з функціональним програмуванням і безпекою [6].

1.1.2.2 Переваги програмування на Kotlin

Kotlin дає змогу писати код компактніше та виразніше, що прискорює процес розробки. Завдяки статичній типізації та функціональним можливостям, Kotlin також сприяє збільшенню продуктивності додатків.

Kotlin надає безліч інструментів для забезпечення безпеки коду. Це дає змогу уникати багатьох помилок на етапі компіляції, що скорочує час і ресурси, що витрачаються на налагодження.

Kotlin сприяє написанню чистішого та читабельнішого коду завдяки своєму синтаксису та функціональним конструкціям. Це робить підтримку і супровід проєкту простішими завданнями.

Kotlin має багату екосистему бібліотек та інструментів, що спрощує розробку та розширення функціональності додатків [5].

1.1.2.3 Перспективи розробки на Kotlin

Одним із ключових аспектів перспектив розробки на Kotlin є активний розвиток самої мови. Нові версії Kotlin регулярно випускаються з поліпшеннями і новими функціями, що робить мову потужнішою і

зручнішою для розробки додатків.

Екосистема Kotlin постійно зростає. З кожним роком з'являються нові бібліотеки, фреймворки та інструменти, розроблені спеціально для Kotlin. Це спрощує розробку додатків і розширює функціональність мови. Kotlin активно використовується в таких галузях, як мобільна розробка (Android), веброзробка, бекенд-розробка, а також ігри.

Kotlin став офіційною мовою розробки для Android. Google активно підтримує Kotlin в Android Studio, надаючи інструменти та ресурси для розробки. Це означає, що Kotlin продовжуватиме домінувати в галузі мобільної розробки на Android, і його популярність зростатиме паралельно зі зростанням популярності платформи Android.

Синтаксис Kotlin «чистий» і легко читається, що спрощує супровід коду. Також Kotlin надає безліч можливостей для написання ефективного і виразного коду.

Kotlin – це потужна і гнучка мова програмування, яка надає безліч інструментів для створення надійних і ефективних додатків. З її допомогою можна розробляти мобільні додатки, серверні додатки та багато іншого. Kotlin є однією з найперспективніших мов програмування [6].

1.1.2.4 Поєднання Jetpack Compose з мовою Kotlin

Під час розробки мобільних додатків для Android мова Kotlin використовується для написання логіки програми, а Jetpack Compose для створення сучасного та декларативного інтерфейсу користувача.

Kotlin і Compose інтегруються між собою безперешкодно, що забезпечує гладку розробку та покращену читабельність коду. Наприклад, в Kotlin можна використовувати Compose-функції для декларативного опису елементів інтерфейсу, таких як кнопки, тексти, та контейнери. Це спрощує розробку UI та забезпечує швидке створення додатків для платформи

Android, використовуючи сучасні підходи у програмуванні.

На відміну від XML, у Compose виконується опис необхідних властивостей у composable-функції та визначають state (поточний стан) елементів, на основі чого система створює та відображає той чи інший інтерфейс на екрані. Елементи додатків у Compose є функціями, а івенти змінюють стани і відповідно сам інтерфейс.

Важливим компонентом Compose є Slot API із резервом місця у контейнері, у якому можна розмістити текст, кнопку, іконки або малюнки без обмежень для розробки. Compose застосовує макети, які за розташуванням схожі до макетів XML, зокрема, Box, Row, Column, LazyColumn. Окрім цього, у Compose використовується макет ConstraintLayout, який надає можливість задавати параметри елементам, а також розташовувати і відображати їх на екрані у бажаному порядку. Макет Scaffold забезпечує розміщення усіх елементів на екрані у звичному місці у Android-додатках [22].

Параметр composable-функції, modifier, описує багато параметрів елементів інтерфейсу, які можна відображати. Існує багато методів його використання з іншими елементами інтерфейсу. При цьому у разі вказівки невідповідних властивостей до елементів інтерфейсу modifier не дозволить це зробити, якщо певний елемент не підтримує ту чи іншу властивість.

Використання Compose для створення інтерфейсу користувача є дуже ефективним та менш енергозатратним. Так, під час побудови інтерфейсу за допомогою XML треба створити клас Kotlin, а також XML-файл та поєднати їх. Після цього потрібно вказати елементи інтерфейсу, описати властивості та поєднати ці елементи з View-моделями та іншими ресурсами, а також задати характеристики для бажаного відображення.

Застосовуючи Compose, потрібно лише створити клас Kotlin, описати функції з відображенням і передати туди всі дані із характеристиками. Compose значно полегшує та пришвидшує цей процес, оскільки з ним зникає потреба створювати XML-файли, які зазвичай є достатньо великими.

Compose надає можливість застосовувати усі функції Kotlin, при цьому використовуючи менше коду.

Режим Preview у Compose є дуже зручним, так як на будь-яку довільну composable-функцію можна написати Preview-функцію і одразу подивитись, як вона виглядає. Також Jetpack Compose дозволяє комбінувати різні composable-функції на різних макетах із різними параметрами, що неможливо зробити в XML [17].

1.2 Платформа Flutter та мова програмування Dart

Платформа Flutter з відкритим вихідним кодом, який розроблено та підтримується Google. Використовується для створення інтерфейсу додатків (UI) для різних платформ із застосуванням єдиної бази коду. Flutter підтримує розробку програм на таких платформах, як iOS, Android, вебінтерфейс, Windows, MacOS, Linux та спрощує процес створення одноманітних привабливих інтерфейсів для додатка на цих платформах.

Розробка крос-платформових додатків дає можливість використовувати одну мову програмування та одну базу коду, щоб створити програму для кількох платформ. Якщо випускається програма для декількох платформ, крос-платформова розробка вимагатиме менших витрат і меншого часу, ніж розробка платформозалежної програми, це дозволяє створювати однакові інтерфейси для різних платформ.

У Flutter реалізовані можливості, які роблять розробку крос-платформових додатків більш простою та високоефективною.

Flutter не покладається на платформозалежні інструменти відображення, а використовує для відображення інтерфейсу користувача графічну бібліотеку Google Skia з відкритим вихідним кодом. Це надає користувачам однакові візуальні елементи незалежно від платформи, яка використовується для доступу до програми.

Google створила Flutter з акцентом на простоті використання. Завдяки таким функціям, як гаряче перезавантаження, можна попередньо переглядати, як виглядатимуть зміни в коді, без втрати стану. Інші інструменти, такі як інспектор віджетів, спрощують візуалізацію та вирішення проблем у макетах інтерфейсу користувача.

Flutter використовує мову програмування з відкритим кодом Dart, який також розроблений в Google. Dart оптимізовано для створення інтерфейсу користувача, і багато його переваг використовуються у Flutter.

Наприклад, одна з можливостей Dart, яка використовується у Flutter, – захист від нульових посилань. Захист від нульових покажчиків спрощує виявлення поширених помилок. Ця можливість скорочує час, що витрачається на обслуговування коду [21].

1.3 Платформа Xamarin

Xamarin – це платформа для розробки мобільних додатків, яка дозволяє використовувати мову програмування C# та .NET для створення крос-платформових застосунків.

Велика частина коду може бути перевикористана між різними платформами, що спрощує розробку та зменшує зусилля для підтримки. Ця платформа є частиною екосистеми Microsoft і надає зручний інструментарій для створення додатків для Android, iOS та Windows.

Основна перевага Xamarin полягає в можливості розробки крос-платформових додатків. Можна використовувати спільний код для реалізації функціональності та подальшої компіляції для кожної платформи окремо, що забезпечує ефективну розробку та підтримку.

Xamarin інтегрується з інтегрованим середовищем розробки (IDE) Microsoft Visual Studio. Це забезпечує багатофункціональне середовище для написання, відлагодження та тестування коду.

Xamarin дозволяє отримати високий рівень продуктивності та швидкодії, так як додатки компілюються в native код для кожної платформи. Це дозволяє використовувати всі можливості платформ та отримувати природний інтерфейс користувача.

Xamarin.Forms дозволяє створювати крос-платформові додатки з одним спільним кодом та інтерфейсом, але іноді може бути обмеженим в плані кастомізації.

Xamarin.Native надає повний контроль над native елементами інтерфейсу користувача, але вимагає більше коду для підтримки різних платформ [15].

1.4 Платформа React Native (RN)

React Native – це фреймворк для розробки крос-платформових мобільних додатків. Він базується на бібліотеці React і дозволяє використовувати один код для створення додатків для різних платформ, таких як iOS та Android.

В основі React Native лежить використання мови програмування JavaScript та розширення її функціоналу за допомогою JSX. Це дозволяє при розробці ефективно використовувати концепції реактивного програмування для створення мобільних додатків.

Фреймворк підтримує «гаряче перезавантаження», що полегшує внесення змін в код і відстеження їх впливу без перезапуску додатку. React Native також надає можливість використовувати native компоненти для отримання повного контролю над виглядом та поведінкою додатку.

React Native є ефективним рішенням, як простий інструмент для крос-платформової розробки мобільних додатків, який використовує відомі технології – JavaScript і React [2].

1.5 Платформа Firebase

Firebase – це платформа розробки мобільних та вебзастосунків. Firebase розвивається з 2011 року компанією Firebase Inc., яку придбав Google у 2014. Firebase пропонує для інтеграції у проєкти розробки декілька служб та рішень.

Firebase Analytics – безкоштовне рішення для оцінки застосунків, яке дає змогу ознайомитись із використанням застосунків та залученням користувачів [7].

Firebase Cloud Messaging. Раніше відомий як Google Cloud Messaging (GCM), Firebase Cloud Messaging (FCM) – це крос-платформове рішення для повідомлень і нотифікацій для Android, iOS та вебзастосунків, які наразі можна використовувати безкоштовно [9].

Firebase Authentication – це служба, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта. Він підтримує соціальні логін-провайдери Facebook, GitHub, Twitter і Google (і Google Play Games). Крім того, вона включає в себе систему управління користувачами, за допомогою якої розробники можуть увімкнути автентифікацію користувача за допомогою входу з електронної пошти та пароля, що зберігаються в Firebase [8, 19].

Realtime Database. Firebase надає в режимі реального часу базу даних та бекенд, як службу. Ця служба надає розробникам застосунків API, який дозволяє синхронізувати дані застосунків між клієнтами (див. рис. 1.1) та зберігати їх у хмарі Firebase [18, 19]. Компанія також надає клієнтські бібліотеки, які дозволяють інтеграцію із застосунками Android, iOS, JavaScript / Node.js, Java, Objective-C, Swift. База даних також доступна через REST API та прив'язки до декількох сценаріїв JavaScript, таких як AngularJS, React, Ember.js та Backbone.js. REST API використовує протокол подій із сервером, який є інтерфейсом для створення HTTP-з'єднань для отримання push-повідомлень від сервера. Розробники, які використовують Realtime

Database, можуть захищати свої дані за допомогою правил безпеки, що застосовуються на сервері [11]. Cloud Firestore, яка є наступною генерацією Firebase Realtime Database, була випущена у бета-версії.

Firebase Storage забезпечує надійне завантаження та вивантаження файлів для застосунків Firebase незалежно від якості мережі. Розробник може використовувати його для зберігання зображень, аудіо-, відео- чи іншого вмісту, створеного користувачами. Зберігання Firebase підтримується Google Cloud Storage [12, 22].

Firebase Hosting – це статичний та динамічний вебхостинг, який було запущено 13 травня 2014 року. Він підтримує хостинг статичних файлів, таких як CSS, HTML, JavaScript та інші файли, а також динамічну підтримку Node.js через Cloud Functions. Служба передає файли через мережу доставки контенту (CDN) за допомогою протоколу HTTPS та шифрування SSL. Firebase підтримує Fastly, CDN, щоб забезпечити підтримку CDN Firebase Hosting. Компанія стверджує, що хостинг Firebase виріс із запитів клієнтів; розробники використовували Firebase для своєї бази даних в режимі реального часу, але вони потребували місця для розміщення їхнього вмісту [21].

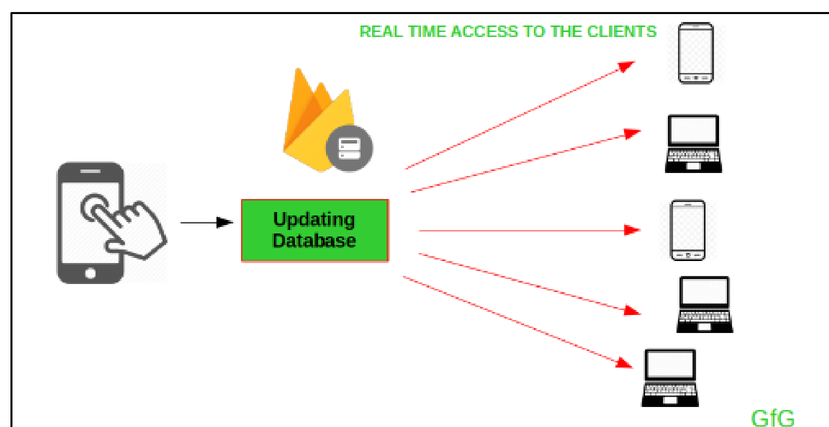


Рисунок 1.1 – Приклад синхронізації даних на пристроях клієнтів [14]

ML Kit – це мультиплатформений мобільний SDK від Google, що надає можливість розробникам легко інтегрувати високоточні наперед навчені

глибинні моделі машинного навчання в мобільні застосунки на Android та iOS [21]. В Android підтримка ML Kit починається з API 21 версії 5.0 (Lollipop). На презентації у 2018 було представлено 5 функцій, сьогодні ж ця кількість збільшилась до 11:

– розпізнавання тексту;
– розпізнавання обличч;
– сканування штрих-кодів;
– маркування зображень;
– виявлення та відстеження об'єктів;
– розпізнавання пам'яток;
– розпізнавання мови тексту;
– переклад;
– генерація «розумних» відповідей;
– створення власних моделей з AutoML;
– задання власних моделей [16].

Перевагою ML Kit є легкість використання: для вирішення більшості завдань досить написати декілька стрічок коду. Для початку роботи з ML Kit не потрібно мати глибинних знань з нейронних мереж та оптимізації моделей. Тим не менш, на випадок, якщо існуючого функціоналу недостатньо для вирішення задачі, ML Kit надає зручну API, яка дозволяє використовувати власні TensorFlow Lite моделі (див. рис. 1.2) в мобільних додатках [20]. Якщо модель занадто складна, можна запустити її тренування в хмарі та організувати динамічне завантаження в додаток. Це дає можливість зменшити розмір додатку і при цьому оновлювати модель окремо від оновлення застосунку [7].

ML Kit забезпечує два режими роботи: на пристрої та в хмарі. Режими можна обирати залежно від існуючих потреб. Тренування моделей на пристрої швидше та не потребує стабільного підключення до інтернету, але хмарні моделі мають більше можливостей і дають вищу точність [3].

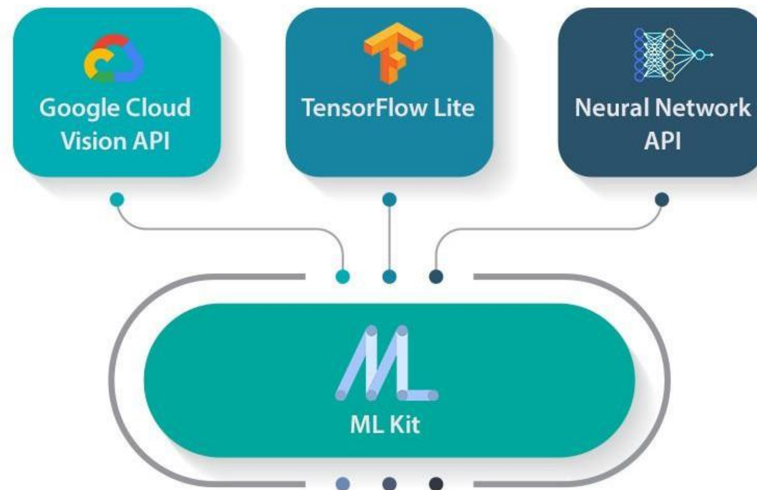


Рисунок 1.2 – Компоненти ML Kit

Також ці два режими мають ще деякі особливості. Розпізнавання пам’яток працює лише в хмарі, а розпізнавання мов та маркування зображень мають більше можливостей ніж при тренуванні на пристрої (більша кількість мов, та об’єктів, які можна розпізнати). На пристрої API пропонує роботу з функціоналом для розпізнавання облич, мови, виявлення та відстеження об’єктів, скануванням QR-кодів, «розумними» відповідями, AutoML та власними кастомними моделями [10].

Важливим аспектом при роботі з хмарними рішеннями є приватність. Тому Google гарантує, що дані користувача видаляються з хмарного API відразу після роботи з ними.

ML Kit також містить ряд недоліків. Зокрема розмір власних моделей часто може перевищувати розмір застосунку в цілому, що створює незручності як для розробника так і для користувача [16].

2 СЕРВІСИ ТА ТЕХНОЛОГІЇ ВИКОРИСТАНІ В ПРОЄКТІ

2.1 Сервіс Firebase Authentication

Firebase Authentication – це сервіс, який надає можливість додавати аутентифікацію користувачів у мобільні та вебдодатки, використовуючи інфраструктуру Firebase. Основна мета – спростити та забезпечити безпеку процесу входу та реєстрації користувачів.

Firebase Authentication підтримує різні способи аутентифікації, такі як електронна пошта та пароль, вхід за допомогою облікових записів Google, Facebook, Twitter, GitHub тощо (див. рис. 2.1).

```
// Приклад коду для аутентифікації за допомогою електронної пошти та пароля
firebase.auth().createUserWithEmailAndPassword(email, password)
  .then((userCredential) => {
    // Реєстрація успішна
    const user = userCredential.user;
  })
  .catch((error) => {
    // Обробка помилок під час реєстрації
    const errorCode = error.code;
    const errorMessage = error.message;
  });

// Приклад коду для входу за допомогою Google
const provider = new firebase.auth.GoogleAuthProvider();

firebase.auth().signInWithPopup(provider)
  .then((result) => {
    // Вхід успішний
    const user = result.user;
  })
  .catch((error) => {
    // Обробка помилок під час входу
    const errorCode = error.code;
    const errorMessage = error.message;
  });
```

Рисунок 2.1 – Приклади використання Firebase Authentication

Firebase Authentication забезпечує безпеку зберігання та передачу облікових даних користувачів, що шифруються та зберігаються в захищеному середовищі Firebase [4].

Firebase Authentication може легко інтегруватися з іншими сервісами Firebase, що дозволяє створювати повністю функціональні та безпечні додатки (див. рис. 2.2).

```
firebase.auth().onAuthStateChanged((user) => {
  if (user) {
    // Користувач увійшов в систему
    console.log("Користувач увійшов:", user.displayName);
  } else {
    // Користувач вийшов з системи
    console.log("Користувач вийшов");
  }
});
```

Рисунок 2.2 – Приклад моніторингу подій входу/виходу

Firebase Authentication надає інструменти для налаштування екрану входу та реєстрації у відповідності до дизайну додатків (див. рис. 2.3).

```
<div id="login-screen">
  <input type="email" id="email-input" placeholder="Електронна пошта">
  <input type="password" id="password-input" placeholder="Пароль">
  <button onclick="performLogin()">Увійти</button>
</div>
```

Рисунок 2.3 – Приклад коду власного екрану входу

Firebase Authentication надає засіб легко відновити забуті паролі користувачів. Це полегшує процес відновлення доступу до облікового запису, а також надає засоби для моніторингу активності користувачів та здійснення аналізу, так щоб було зрозуміло, як користувачі використовують систему аутентифікації.

Firebase Authentication є потужним та зручним сервісом для розробки, якщо потрібне швидке та безпечне рішення для реалізації аутентифікації у додатках.

2.2 Роль та важливість машинного навчання в розробці мобільних додатків

Firebase ML Kit – це набір інструментів для розробки мобільних додатків, який надає доступ до функцій машинного навчання (МН).

Firebase ML Kit складається з ряду модулів, кожен з яких призначений для вирішення конкретних задач МН, таких як розпізнавання тексту, обличчя чи мови. Це дозволяє вибирати лише необхідні функціональності для додатків.

ML Kit підтримує як Android, так і iOS, що робить його ідеальним вибором для розробки крос-платформенних мобільних додатків. Також у ML Kit є можливість використання хмарних технологій для розширення можливостей додатків. Тобто можна використовувати великі та постійно оновлювані моделі, які зберігаються в хмарі.

МН в розробці мобільних додатків стає все більш важливим елементом, оскільки дозволяє створювати інтелектуальні та адаптивні додатки. Додавання функцій МН може значно покращити користувацький досвід та забезпечити нові можливості [20].

МН дозволяє створювати моделі для розпізнавання об'єктів, зображень, текстів та іншого контенту. Наприклад, використання ML Kit для розпізнавання обличчя FaceDetector (див. рис. 2.4) на зображеннях у реальному часі.

МН використовується для аналізу поведінки користувачів та надання персоналізованих рекомендацій. Наприклад, додатки можуть пропонувати користувачам товари чи контент, які є найбільш цікавими для них.

ML Kit використовується для впровадження біометричних методів аутентифікації, таких як розпізнавання обличчя чи відбитків пальців. Це підвищує рівень безпеки додатків та забезпечує зручну аутентифікацію.

Завдяки МН, додатки можуть отримувати нові можливості, такі як розпізнавання мови, автоматичний переклад, аналіз настроїв та інші. Це

дозволяє створювати додатки, які взаємодіють з користувачами більш інтелектуально.

```kotlin
// Приклад використання Face Detector в Kotlin
val options = FirebaseVisionFaceDetectorOptions.Builder()
    .setModeType(FirebaseVisionFaceDetectorOptions.ACCURATE_MODE)
    .build()
val detector = FirebaseVision.getInstance().getVisionFaceDetector(options)
val image = FirebaseVisionImage.fromBitmap(myBitmap)
detector.detectInImage(image)
    .addOnSuccessListener { faces ->
        // Обробка результатів розпізнавання облич
    }
    .addOnFailureListener { e ->
        // Обробка помилок
    }
```

Рисунок 2.4 – Приклад використання FaceDetector

Використання таких сервісів, як Authentication, FaceDetector та інших сервісів, може значно покращити функціональність та забезпечити нові перспективи у розробці мобільних додатків на мові Kotlin [19].

2.3 ML Kit FaceDetector

2.3.1 Концепції розпізнавання обличчя

Сервіс визначає людські обличчя в режимі реального часу або на візуальних носіях, таких як цифрові зображення або відео. Коли обличчя виявлено, йому призначають відповідне положення, розмір і орієнтацію; його можна відстежувати за ознаками, такими як очі та ніс.

Ось деякі терміни, які використовуються для роботи функції виявлення обличчя в ML Kit – відстеження обличчя, орієнтир, контур та класифікація.

Відстеження обличчя розширює розпізнавання обличчя на відео. Будь-яке обличчя, яке з'являється на відео протягом будь-якого проміжку часу, можна відстежувати від кадру до кадру. Це означає, що обличчя, виявлене в послідовних відеокадрах, може бути ідентифіковано як одне й те ж обличчя.

Відстеження обличчя робить висновки лише на основі положення та руху обличчя в відеопотоці.

Орієнтир – це визначена точка всередині обличчя. Ліве око, праве око і основа носа – це є прикладами орієнтирів. ML Kit надає можливість знаходження орієнтирів на виявленому обличчі.

Контур – це набір точок, які повторюють форму обличчя. ML Kit надає можливість знаходження контурів обличчя. Класифікація визначає, чи присутня певна ознака обличчя. Наприклад, обличчя можна класифікувати за тим, відкриті чи закриті очі, чи є посмішка на обличчі.

2.3.2 Орієнтація, контури, класифікація та мінімальний розмір обличчя

Наступні терміни описують кут, під яким обличчя орієнтоване відносно камери:

– Ейлер X : грань з додатним кутом Ейлера X, яка повернута вгору;
– Ейлер Y : обличчя з додатним кутом Ейлера Y дивиться вправо від камери або вліво, якщо він від'ємний;
– Ейлер Z : обличчя з додатним кутом Ейлера Z обертається проти годинникової стрілки відносно камери.

ML Kit визнає обличчя, не шукаючи орієнтирів. Виявлення орієнтирів – необов'язковий етап, який за замовчуванням вимкнено.

У таблиці 2.1 подані всі орієнтири, які можна виявити з урахуванням кута Ейлера Y пов'язаної грані.

Таблиця 2.1 – Залежність виявлених орієнтирів від кута Ейлера Y

Кут Ейлера Y	Виявлені орієнтири
< -36 градусів	ліве око, ліва частина рота, ліве вухо, основа носа, ліва щока

Продовження табл. 2.1

Кут Ейлера Y	Виявлені орієнтири
від -36 градусів до -12 градусів	ліва частина рота, основа носа, нижня частина губи, праве око, ліве око, ліва щока, ліве вухо
-12 градусів до 12 градусів	праве око, ліве око, основа носа, ліва щока, права щока, ліва частина рота, права частина рота, нижня губа
від 12 градусів до 36 градусів	права частина рота, основа носа, нижня частина губи, ліве око, праве око, права щока, праве вухо
>36 градусів	праве око, права частина рота, праве вухо, основа носа, права щока

Кожний виявлений орієнтир має пов'язане з ним положення на зображенні.

Контур – це набір точок, які представляють форму та риси обличчя (див. рис. 2.5).

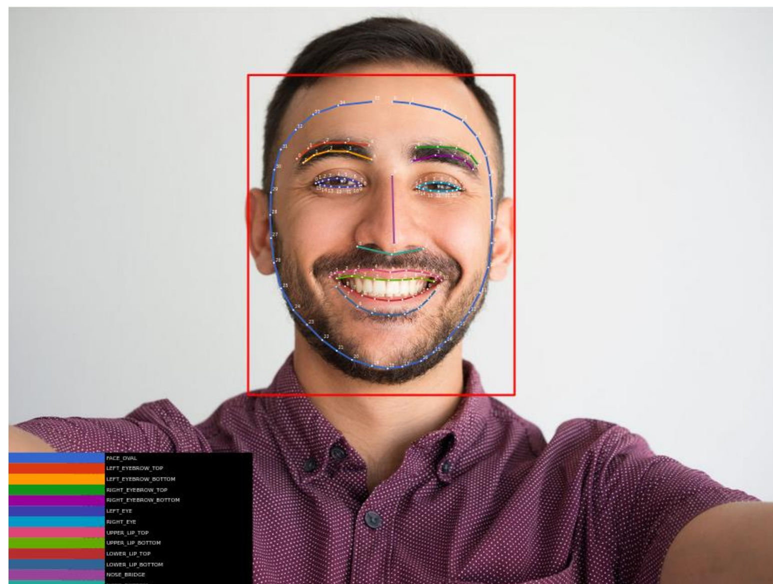


Рисунок 2.5 – Розподіл точок при ідентифікації обличчя

Кожний контур об'єкта, який виявляє ML Kit, представлений фіксованою кількістю точок (див. табл. 2.2).

Таблиця 2.2 – Розподіл контрольних точок для контурів обличчя

Овал обличчя	36 точок	Верхня губа (верхня лінія)	11 точок
Ліва брова (верхня лінія)	5 точок	Верхня губа (нижня лінія)	9 точок
Ліва брова (нижня лінія)	5 точок	Нижня губа (верхня лінія)	9 точок
Права брова (верхня лінія)	5 точок	Нижня губа (нижня лінія)	9 точок
Права брова (нижня лінія)	5 точок	Перенісся	2 точки
Ліва щока (центр)	1 точка	Нижня частина носа	3 точки
Права щока (центр)	1 точка		

Коли отримуємо всі контури обличчя одночасно, то це є масив зі 133 точок, які відповідають контурам об'єктів, як показано в таблиці 2.3.

Таблиця 2.3 – Розподіл контрольних точок в індексах масиву обробки

Індекси контурів об'єктів	Частина контуру обличчя
0-35	Овал обличчя
36-40	Ліва брова (верхня лінія)
41-45	Ліва брова (нижня лінія)
46-50	Права брова (верхня лінія)
51-55	Права брова (нижня лінія)
56-71	Ліве око
72-87	Праве око
88-96	Верхня губа (нижня лінія)
97-105	Нижня губа (верхня лінія)
106-116	Верхня губа (верхня лінія)
117-125	Нижня губа (нижня лінія)

Продовження табл. 2.3

Індекси контурів об'єктів	Частина контуру обличчя
126-127	Перенісся
128-130	Нижня частина носа
131	Ліва щока (центр)
132	Права щока (центр)

Класифікація визначає, чи присутня певна ознака обличчя. ML Kit наразі підтримує дві класифікації: очі відкриті/закриті або посмішка (є/ні).

Класифікація є показником достовірності. Вона вказує на впевненість у наявності певної характеристики обличчя. Наприклад, значення 0,7 або більше для класифікації посмішки вказує на те, що людина, ймовірно, посміхається. Обидві ці класифікації базуються на виявленні орієнтирів. Також ці класифікації працюють лише для фронтальних облич, тобто облич із невеликим кутом Ейлера Y (від -18 до 18 градусів).

Мінімальний розмір обличчя не є жорстким обмеженням для роботи детектора (див. рис. 2.6).

Рисунок 2.6 – Розпізнавання кількох облич та їх визначення

Мінімальний розмір обличчя – це співвідношення ширини голови до ширини зображення. Наприклад, значення 0,1 означає, що найменше обличчя для пошуку займає приблизно 10% ширини шуканого зображення. Це також компроміс між продуктивністю та точністю. Встановлення меншого мінімального розміру дозволяє детектору знаходити менші обличчя, але виявлення займе більше часу, а встановлення більшого розміру може виключити обличчя дрібного розміру, але буде працювати швидше.

2.3.3 Можливості та технічні аспекти розпізнавання обличчя

Firebase ML Kit FaceDetector – потужний інструмент для розпізнавання обличчя в реальному часі у мобільних додатках, що працює на базі технологій МН (див. рис. 2.7).

```
// Ініціалізація Face Detector
val options = FirebaseVisionFaceDetectorOptions.Builder()
    .setPerformanceMode(FirebaseVisionFaceDetectorOptions.ACCURATE)
    .setLandmarkMode(FirebaseVisionFaceDetectorOptions.ALL_LANDMARKS)
    .setClassificationMode(FirebaseVisionFaceDetectorOptions.ALL_CLASSIFICATIONS)
    .setMinFaceSize(0.15f)
    .enableTracking()
    .build()

val faceDetector = FirebaseVision.getInstance().getVisionFaceDetector(options)

// Обробка фото
val image: FirebaseVisionImage = FirebaseVisionImage.fromBitmap(myBitmap)

// Запуск Face Detector
faceDetector.detectInImage(image)
    .addOnSuccessListener { faces ->
        // Обробка результатів розпізнавання
        for (face in faces) {
            val bounds = face.boundingBox
            val rotY = face.headEulerAngleY // Поворот по Y
            val rotZ = face.headEulerAngleZ // Поворот по Z

            // Додаткові дані, такі як емоції, можливості обличчя та ін.
        }
    }
    .addOnFailureListener { e ->
        // Обробка помилок
    }
```

Рисунок 2.7 – Лістинг коду Face Detector

FaceDetector визначає ключові точки обличчя, такі як очі, ніс та рот, що дозволяє визначити положення та вираз обличчя. Інтегрована система може визначати емоційний стан користувача на основі виразів обличчя.

FaceDetector використовує вбудовані моделі МН, що дозволяє миттєво використовувати його можливості без необхідності навчання власних моделей. Це спрощує розробку та розгортання функціоналу розпізнавання обличчя в додатках.

2.3.4 Розробка додатків для розпізнавання обличчя

FaceDetector може бути використаний для створення додатків, які реагують на обличчя користувача. Наприклад, реалізація системи автоматичного фокусування на камеру або розблокування за допомогою розпізнавання обличчя (див. рис. 2.8).

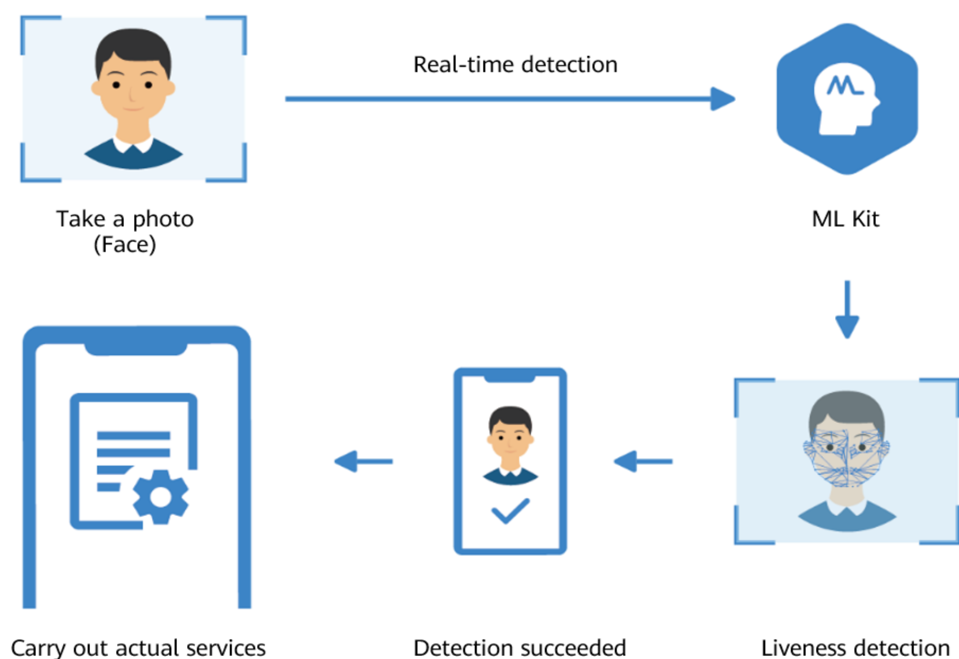


Рисунок 2.8 – Ідентифікація користувача

Приклад використання для автоматичного фокусування камери (див. рис. 2.9).

```
// Розпізнавання обличчя
faceDetector.detectInImage(image)
    .addOnSuccessListener { faces ->
        // Перевірка чи знайдено обличчя
        if (faces.isNotEmpty()) {
            // Автоматичне фокусування камери на обличчі
            camera.autoFocus { success, camera ->
                // Логіка після фокусування
            }
        }
    }
    .addOnFailureListener { e ->
        // Обробка помилок
    }
```

Рисунок 2.9 – Автоматичне фокусування камери

FaceDetector може використовуватися для додавання спеціальних ефектів, наприклад, таких як «маски» при роботі з камерою або у соціальних мережах (див. рис. 2.10).

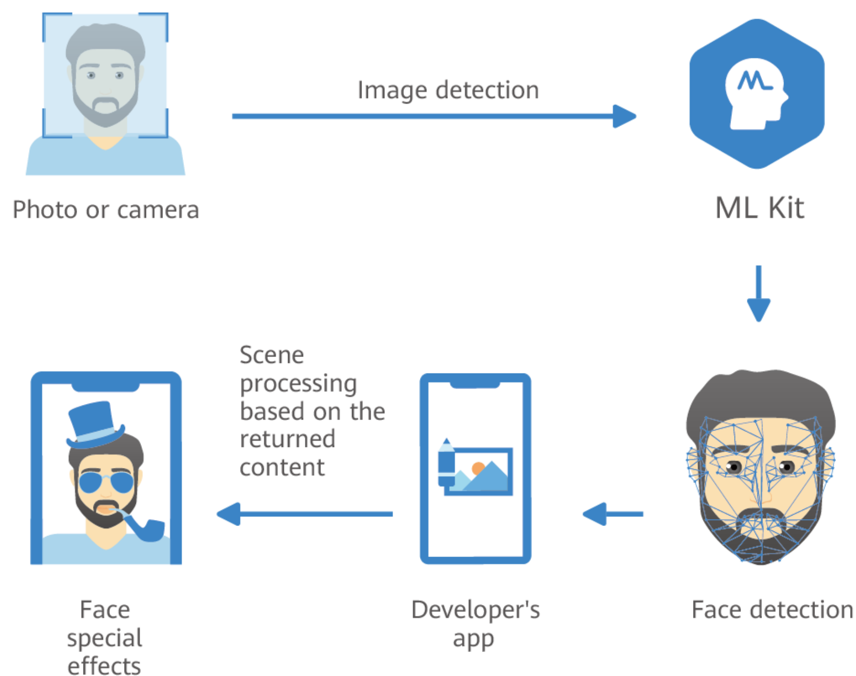


Рисунок 2.10 – Визначення користувача та застосування «маски»

2.3.5 Порівняння з іншими сервісами

Розглянемо ще декілька сервісів, які пропонують схожі можливості розпізнавання обличчя.

OpenCV (Open Source Computer Vision Library) – це бібліотека з відкритим вихідним кодом, яка має реалізації багатьох алгоритмів комп'ютерного зору, включаючи розпізнавання обличчя. Це дозволяє створювати індивідуальне рішення для обробки обличчя за допомогою різних методів.

Microsoft Azure Face API, надає можливості розпізнавання обличчя, включаючи визначення основних атрибутів обличчя, виявлення виразів обличчя, розпізнавання віку і статі та інші функції.

Amazon Rekognition – це сервіс для обробки зображень та відео в хмарі Amazon Webservices. Він має функціонал для розпізнавання обличчя, включаючи виявлення та аналіз основних характеристик обличчя.

Kairos – це компанія, яка спеціалізується на технологіях розпізнавання обличчя. Вони пропонують API для розпізнавання обличчя з різними функціями, такими як виявлення виразів обличчя, визначення статі та віку тощо.

IBM Watson Visual Recognition, також має можливості розпізнавання обличчя. Сервіс може визначати основні атрибути обличчя та виконувати інші завдання розпізнавання.

Порівняльний аналіз сервісів розпізнавання обличчя, таких як ML Kit FaceDetector та OpenCV, результати представлено у таблиці 2.4.

Таблиця 2.4 – Порівняльний аналіз між ML Kit FaceDetector та OpenCV для розпізнавання обличчя

Критерій	Сервіс	
	ML Kit FaceDetector	OpenCV
1. Відкритий вихідний код	ML Kit Face Detector також є частиною платформи Firebase, і його API використовується в	OpenCV є відкритою бібліотекою з відкритим вихідним кодом. Це означає, що ви можете

Продовження табл. 2.4

Критерій	Сервіс	
	ML Kit FaceDetector	OpenCV
	хмарному середовищі, але вихідний код не є відкритим.	вільно використовувати, змінювати та поширювати його код.
2. Інтеграція та зручність використання	ML Kit Face Detector забезпечує просту інтеграцію через хмарний API. Він зручний для використання в мобільних додатках, особливо для розробників, які працюють з платформою Firebase.	OpenCV надає розширений набір функцій для обробки зображень та відео. Він доступний на багатьох платформах, включаючи Android, iOS, Windows, Linux та інші. Також він має широке спільноту та документацію.
3. Продуктивність	Продуктивність ML Kit Face Detector може залежати від швидкості вашого інтернет-з'єднання, оскільки він працює в хмарі. У випадку офлайн-розпізнавання результати також можуть залежати від способу використання моделі на пристрої.	OpenCV часто відомий своєю високою продуктивністю. Його можна оптимізувати та використовувати для вирішення широкого спектру завдань в реальному часі.
4. Моделі та навчання	ML Kit Face Detector використовує моделі глибокого навчання, але деталі щодо навчання та	OpenCV надає інструменти для обробки зображень, але в більшості випадків не навчає свої власні моделі.

Продовження табл. 2.4

Критерій	Сервіс	
	ML Kit FaceDetector	OpenCV
4. Моделі та навчання	структури моделей не завжди доступні.	Замість цього розробники можуть навчати моделі на своїх даних та використовувати їх з OpenCV.

Отже, якщо при розробці потрібна швидка та проста реалізація розпізнавання облич у мобільних додатках з можливістю інтеграції сервісів Firebase, ML Kit буде оптимальним варіантом. У випадках, коли потрібно більше контролю та гнучкості при реалізації процесу розпізнавання рис або міміки обличчя, а також для складних завдань комп'ютерного зору, кращим вибором може бути OpenCV.

Firebase ML Kit FaceDetector відкриває нові можливості для розпізнавання обличчя в мобільних додатках. Використання вбудованих функцій МН дозволяє створювати додатки з покращеним користувацьким інтерфейсом.

З приростом обчислювальної потужності мобільних пристроїв та поширенням використання розпізнавання обличчя в різних сферах, можна очікувати подальше вдосконалення та розширення можливостей Firebase ML Kit FaceDetector.

Таким чином, Firebase ML Kit ставить собі за мету полегшити і прискорити розробку мобільних додатків, надаючи готові інструменти для роботи з аутентифікацією та МН. Застосування цих сервісів може значно поліпшити якість, функціональність та безпеку додатків, забезпечуючи їхню конкурентоспроможність на ринку мобільних застосунків.

3 РОЗРОБКА ANDROID ЗАСТОСУНКУ ЗАСОБАМИ KOTLIN ТА FIREBASE ML KIT

3.1 Створення проєкту та його налаштування

У процесі створення проєкту застосунку використано сучасне IDE Android Studio, яке є основним та рекомендованим інструментом для розробки Android-застосунків, і особливо для розробки на мові Kotlin. Проєкт отримав назву "MyProjectQW", а для його налаштування та оптимізації імплементовано необхідні бібліотеки для розробки на Kotlin (див. рис. 3.1), для роботи з бібліотекою Koin (див. рис. 3.2), та роботи з камерою (див. рис. 3.3).

Застосунок розроблено для тестування користувача в будь-якій галузі знань або навичок – тестову частину коду можна змінювати за потреби.

Для проєкту обрано тест на уважність. Користувачеві пропонується за 60 секунд набрати максимально можливу кількість балів, визначаючи правильний колір тексту. Складність тесту полягає в тому, що і назва кольору і сам колір відображаються у вигляді назв кольорів, що створює виклик для уваги та концентрації.

```
implementation("androidx.room:room-common:2.6.1")
implementation("androidx.core:core-ktx:1.12.0")
implementation("androidx.appcompat:appcompat:1.6.1")
implementation("com.google.android.material:material:1.10.0")
implementation("androidx.constraintlayout:constraintlayout:2.1.4")
implementation("androidx.lifecycle:lifecycle-extensions:2.2.0")
implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.2")
implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2")
implementation("androidx.activity:activity-ktx:1.8.1")
implementation("org.jetbrains.kotlin:kotlin-stdlib:1.9.21")
implementation("com.google.firebase:firebase-auth:22.3.0")
```

Рисунок 3.1 – Стандартні бібліотеки та Kotlin

```
// Koin
api ("io.insert-koin:koin-core:3.1.4")
api ("io.insert-koin:koin-android:3.1.4")
```

Рисунок 3.2 – Бібліотеки Koin

```
// CameraX
val camerax_version = "1.3.0"
implementation("androidx.camera:camera-core:$camerax_version")
implementation("androidx.camera:camera-camera2:$camerax_version")
implementation("androidx.camera:camera-extensions:$camerax_version")
implementation("androidx.camera:camera-lifecycle:$camerax_version")
implementation("androidx.camera:camera-video:$camerax_version")
implementation("androidx.camera:camera-extensions:$camerax_version")
implementation("androidx.camera:camera-view:$camerax_version")
```

Рисунок 3.3 – Бібліотеки для роботи з камерою

3.2 Інтеграція сервісу Firebase Authentication в проєкт

3.2.1 Розробка коду для інтеграції сервісу Firebase Authentication

Інтеграція сервісу Firebase Authentication до складу проєкту забезпечує безпеку та ідентифікацію користувачів. Створено клас LoginActivity, який відповідає за авторизацію, та клас RegistrationActivity, який відповідає за реєстрацію. Вони будуть взаємодіяти з сервісом Firebase Authentication для забезпечення автентифікації користувачів.

Для підключення сервісу в меню IDE Android Studio вибрано пункт "Tools/Firebase/Authentication" та обрано посилання "Authentication using Google". Це викликало перехід на сторінку Firebase Console.

На сторінці Firebase Console створено новий проєкт з назвою "MyProjectQW". Далі, в розділі сервісів, обрано Authentication та підключено його до проєкту. Для ідентифікації користувача обрано пару Email/password та завантажено до кореневої теки проєкту файл google-service.json.

До проєкту також було імплементовано бібліотеку для коректної роботи з Firebase Authentication (див. рис. 3.4).

```
implementation("com.google.firebase:firebase-auth:22.3.0")
```

Рисунок 3.4 – Бібліотека для роботи з Firebase Authentication

Після успішного підключення до сервісу Firebase, додаємо відповідний код для авторизації до класу LoginActivity (див. рис. 3.5, а) та код для реєстрації користувачів до класу RegistrationActivity (див. рис. 3.5, б).

```
if (email_val.isNullOrEmpty() || password_val.isNullOrEmpty()) {
    // Виводимо повідомлення про некоректне введення даних
    Toast.makeText(
        applicationContext,
         text: "Будь ласка, заповніть всі поля",
        Toast.LENGTH_LONG
    ).show()
} else {
    mAuth?.signInWithEmailAndPassword(email_val!!, password_val!!)
        ?.addOnCompleteListener(this@LoginActivity) { task ->
            if (task.isSuccessful) {
                val user = mAuth?.currentUser
                checklogin?.text = "True"
                updateUI(user)
            } else {
                checklogin?.text = "False"
                email?.text?.clear()
                password?.text?.clear()
                Toast.makeText(
                    applicationContext,
                     text: "Невдала авторизація.",
                    Toast.LENGTH_LONG
                ).show()
```

а)

```
if (email.isNullOrBlank() || password.isNullOrBlank()) {
    Toast.makeText(
         context: this,
         text: "Будь ласка, заповніть всі поля",
        Toast.LENGTH_SHORT).show()
} else {
    // Реєстрація нового користувача
    auth.createUserWithEmailAndPassword(email!!, password!!)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Реєстрація успішна
                Log.d(TAG,  msg: "createUserWithEmail:success")
                Toast.makeText(
                     context: this,
                     text: "Ви успішно зареєструвались в системі",
                    Toast.LENGTH_SHORT).show()
                navigateToMainActivity()
            } else {
                // Реєстрація не вдалась
```

б)

Рисунок 3.5 – Лістинги підключення до Firebase Authentication

Розглянемо, які процеси виконує код у цих двох класах.

Робота коду в класі LoginActivity. Код перевіряє, чи обидва обов'язкових поля (електронна пошта і пароль) не є порожніми або null. Якщо хоча б одне з полів порожнє, виводиться повідомлення про необхідність заповнення обох полів.

Використовуючи Firebase Authentication, код викликає метод signInWithEmailAndPassword для авторизації користувача. Результат авторизації обробляється через addOnCompleteListener. Якщо авторизація

успішна, встановлюється відповідне текстове значення checklogin, і викликається функція updateUI для оновлення інтерфейсу користувача, але поки не розроблена наступна частина проєкту, залишаємо цю функцію порожньою. У випадку невдачі авторизації, текстове значення checklogin встановлюється як "False", і поля для введення даних очищаються, а користувач отримує повідомлення про невдалу авторизацію.

У цьому класі передбачена можливість реєстрації користувача, якщо він ще не зареєстрований в системі, за допомогою активного елементу інтерфейсу signup. Коли користувач натискає на цей елемент, викликається блок коду всередині функції setOnClickListener. В цьому блоці коду створюється новий інтент, який призначений для переходу на активність RegistrationActivity (див. рис. 3.6). Об'єкт Intent визначає, яку дію потрібно виконати. У цьому випадку вказується, що потрібно перейти на активність RegistrationActivity. Після створення Intent викликається метод startActivity, який запускає активність RegistrationActivity.

```
// Перехід на вікно реєстрації
signup?.setOnClickListener { it: View!
    val registerIntent = Intent(applicationContext, RegistrationActivity::class.java)
    startActivity(registerIntent)
}
```

Рисунок 3.6 – Обробник переходу до RegistrationActivity

Робота коду в класі RegistrationActivity. Зчитування введених користувачем даних (e-mail і пароль) з відповідних полів введення.

Виконується перевірка, чи введені обидва обов'язкові поля (e-mail і пароль). Якщо хоча б одне з полів порожнє, виводиться коротке сповіщення про необхідність заповнення всіх полів.

Використовуючи Firebase Authentication, код викликає метод createUserWithEmailAndPassword, який реєструє нового користувача за введеними електронною поштою і паролем. Результат реєстрації

обробляється через addOnCompleteListener. Якщо реєстрація успішна, виводиться повідомлення, і викликається функція navigateToMainActivity для оновлення інтерфейсу користувача, але поки не розроблена наступна частина проєкту, залишаємо цю функцію порожньою. У випадку невдачі реєстрації, обробляється відповідна помилка, і користувач отримує відповідне повідомлення разом з очищенням полів введення.

На цьому робота сервісу Firebase Authentication та класів LoginActivity і RegistrationActivity завершується, а користувач перенаправляється до наступної активності MainActivity.

3.2.2 Тестування розробленого коду

Зробимо тестову перевірку на працездатність цієї частини коду. Для цього будемо використовувати вбудований до IDE Android Studio емулятор мобільного пристрою. В налаштуваннях Device Manager в якості пристрою оберемо Pixel_3a_API_34_extension_level_7_x86_64 та інсталюємо його до нашого SDK. Емулятор готовий до роботи, запускаємо проєкт на виконання. Запуск проєкту пройшов без помилок (див. рис. 3.7, а).

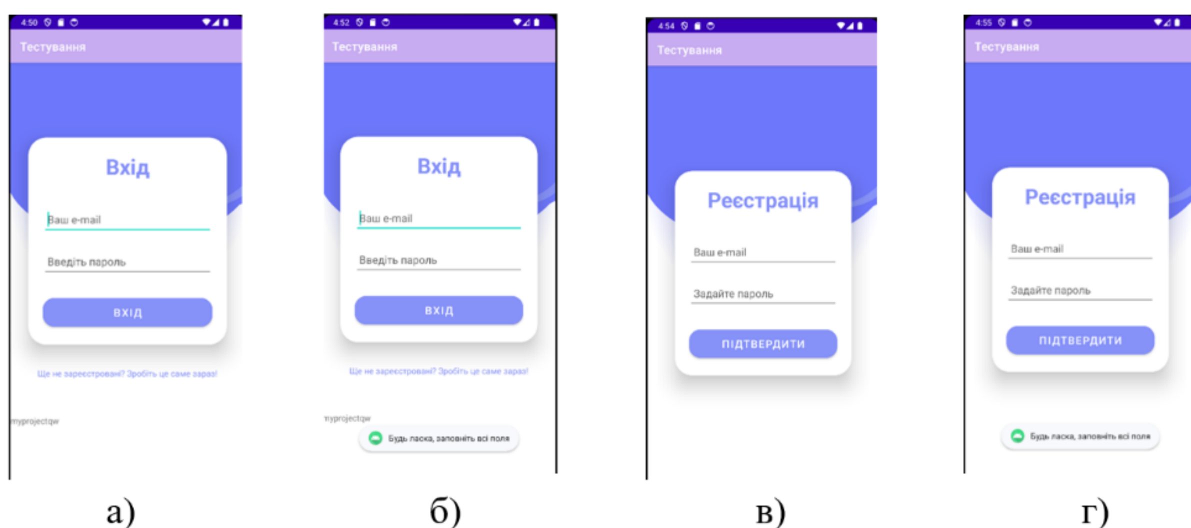


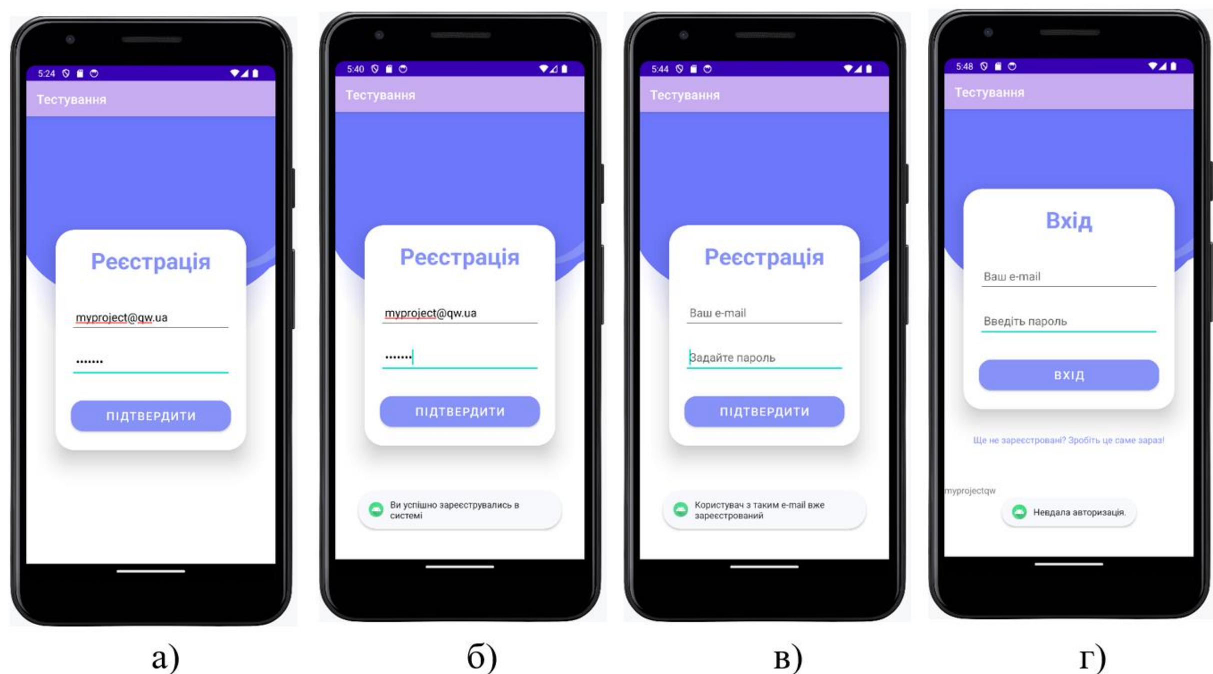
Рисунок 3.7 – Перевірка коду на спробу ввести «порожнє» значення

Зробимо спробу вводу порожніх полів (див. рис. 3.7, б). Так як в проєкті ще немає жодного зареєстрованого акаунту, то спробуємо його зареєструвати (див. рис. 3.7, в), і отримуємо результат на спробу вводу порожніх полів (див. рис. 3.7, г).

Тепер перевіримо, як працює сам сервіс реєстрації, для цього вводимо довільні значення e-mail/password (див. рис. 3.8, а) та отримуємо результат (див. рис. 3.8, б).

При спробі зареєструвати той самий акаунт отримуємо повідомлення (див. рис. 3.8, в).

Перевіримо, яка буде реакція якщо при авторизації ввести правильний e-mail та неправильний password (див. рис. 3.8, г).



а) б) в) г)

Рисунок 3.8 – Перевірка коду на спробу реєстрації

Станом на цей момент зробити перевірку на переходи до активності MainActivity не можливо, це буде перевірено після розробки частини проєкту, яка відповідає за тестування, а ця частина коду дуже вдало пройшла перевірку на працездатність.

3.3 Основний функціонал проєкту

3.3.1Розробка коду для реалізації тесту

Для реалізації частини проєкту, яка відповідає за функціонування тесту зробимо ще декілька активіті.

MainActivity, яка буде проміжною ланкою між процесом авторизації або реєстрації та процесом тестування. В її складі буде дві кнопки: «Почати» та «Вихід», які виконують відповідні обробки подій (див. рис. 3.9).

```
private fun addListenerOnButton() {
    val startBtn = findViewById<View>(R.id.start_btn) as Button
    val exitBtn = findViewById<View>(R.id.exit_btn) as Button
    startBtn.setOnClickListener { it: View!
        val mainIntent = Intent( packageContext: this, TestActivity::class.java)
        startActivity(mainIntent)
        finish()
    }

    exitBtn.setOnClickListener { it: View!
        FirebaseAuth.getInstance().signOut()
        finishAffinity()
    }
}
```

Рисунок 3.9 – Обробник кнопок класу MainActivity

TestActivity, яка реалізує логіку всього процесу тестування. Розглянемо деякі блоки коду окремо.

У методі onCreate відбувається ініціалізація екрану, завантажується розмітка, ініціалізуються різні змінні та встановлюється лічильник часу CountDownTimer для обмеження часу тесту до 60 секунд (див. рис. 3.10). Метод onFinish викликається при завершенні таймера. Після цього створюється новий інтент для переходу до основного коду активності TestActivity із передачею результатів тестування (кількість набраних балів).

```
time = findViewById<View>(R.id.time) as TextView

object : CountDownTimer( millisInFuture: 60000,  countDownInterval: 1000) {
    @SuppressLint("SetTextI18n")
    override fun onTick(l: Long) {
        if (l < 10000) time!!.text = "00:0" + (l / 1000).toString()
        else time!!.text = "00:" + (l / 1000).toString()
    }

    override fun onFinish() {
        time!!.text = "00:00"

        val intent = createIntent( context: this@MainActivity, points!!.text)
        finish()
        startActivity(intent)
    }
}.start()
```

Рисунок 3.10 – Реалізація таймеру

У методах onYesClick (див. рис. 3.11) та onNoClick встановлюються слухачі для кнопок «Так» та «Ні», які викликають відповідні методи при натисканні. Ці методи здійснюють логіку тесту, перевіряючи правильність вибору кольорів та збільшуючи кількість балів користувача.

```
private fun onYesClick() {
    val yes = findViewById<View>(R.id.yes_btn) as Button
    points = findViewById<View>(R.id.points) as TextView
    color1 = findViewById<View>(R.id.color1) as TextView
    color2 = findViewById<View>(R.id.color2) as TextView
    yes.setOnClickListener { it: View!
        val c2 = color2?.currentTextColor
        if (c2 == dictionary[color1?.text.toString()]) {
            val prev = points?.text.toString().toIntOrNull() ?: 0
            points?.text = (prev + 1).toString()
        }
        changeColors()
        attempt += 1
    }
}
```

Рисунок 3.11 – Реалізація методу onYesClick

Методи changeColors, randomChangeColors (див. рис. 3.12), і fixedChangeColors відповідають за зміну кольорів, які відображаються на

екрані під час тестування. Метод changeColors визначає, чи використовувати випадковий вибір кольорів чи фіксований порядок.

```
private fun randomChangeColors() {
    val r = Random()
    color1!!.text = keyList!![r.nextInt(size)]
    color1!!.setTextColor((values[r.nextInt(values.size)] as Int))
    color2!!.text = keyList!![r.nextInt(size)]
    color2!!.setTextColor((values[r.nextInt(values.size)] as Int))
}
```

Рисунок 3.12 – Реалізація методу randomChangeColors

Об'єкт companion object містить статичні методи, які можна викликати без створення екземпляра класу. Наприклад, метод createIntent створює інтент для переходу до ResultActivity, передаючи кількість балів (див. рис. 3.13).

```
companion object {
    fun createIntent(context: Context?, points: CharSequence?): Intent {
        val i = Intent(context, ResultActivity::class.java)
        i.putExtra( name: "result", points)
        return i
    }
```

Рисунок 3.13 – Реалізація передачі результатів

ResultActivity, яка буде виводити результат тесту на екран, та пропонувати зробити вибір, при натискані на відповідні кнопки: «Спробувати ще» та «На головну» (див. рис. 3.14). У методі onCreate, який викликається при створенні активності, отримується посилання на TextView за його ідентифікатором, і до текстового поля result додається значення, передане з TestActivity через Intent під ключем "result". У методі addListenerOnButton встановлюються слухачі натискань на кнопки play_btn і

main_btn. При натисканні на кнопку play_btn, створюється інтент для переходу на TestActivity, поточна активність закривається, і запускається нова активність. Аналогічно, для кнопки main_btn створюється інтент для переходу на MainActivity.

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_result)
    result = findViewById<View>(R.id.result) as TextView
    addListenerOnButton()
    val intent = intent
    result!!.text = intent.getStringExtra( name: "result")
}

private fun addListenerOnButton() {
    play_btn = findViewById<View>(R.id.play_btn) as Button
    main_btn = findViewById<View>(R.id.main_btn) as Button
    play_btn!!.setOnClickListener { it: View!
        val intentMain = Intent( packageContext: this@ResultActivity, TestActivity::class.java)
        finish()
        startActivity(intentMain)
    }
    main_btn!!.setOnClickListener { it: View!
        val intentStart = Intent( packageContext: this@ResultActivity, MainActivity::class.java)
        finish()
        startActivity(intentStart)
    }
}
```

Рисунок 3.15 – Лістинг коду класу ResultActivity

Тепер додамо функції для класів LoginActivity (див. рис. 3.15, а) та RegistrationActivity (див. рис. 3.15, б) на здійснення переходу до MainActivity.

```kotlin
private fun updateUI(currentUser: FirebaseUser?) {
    val profileIntent = Intent(applicationContext, MainActivity::class.java)
    profileIntent.putExtra( name: "email", currentUser?.email)
    currentUser?.uid?.let { Log.v( tag: "DATA", it) }
    startActivity(profileIntent)
}
```

а)

```kotlin
private fun navigateToMainActivity() {
    // Перехід до активіті MainActivity
    val intent = Intent( packageContext: this, MainActivity::class.java)
    startActivity(intent)
    finish() // Закриваємо поточну активіті
}
```

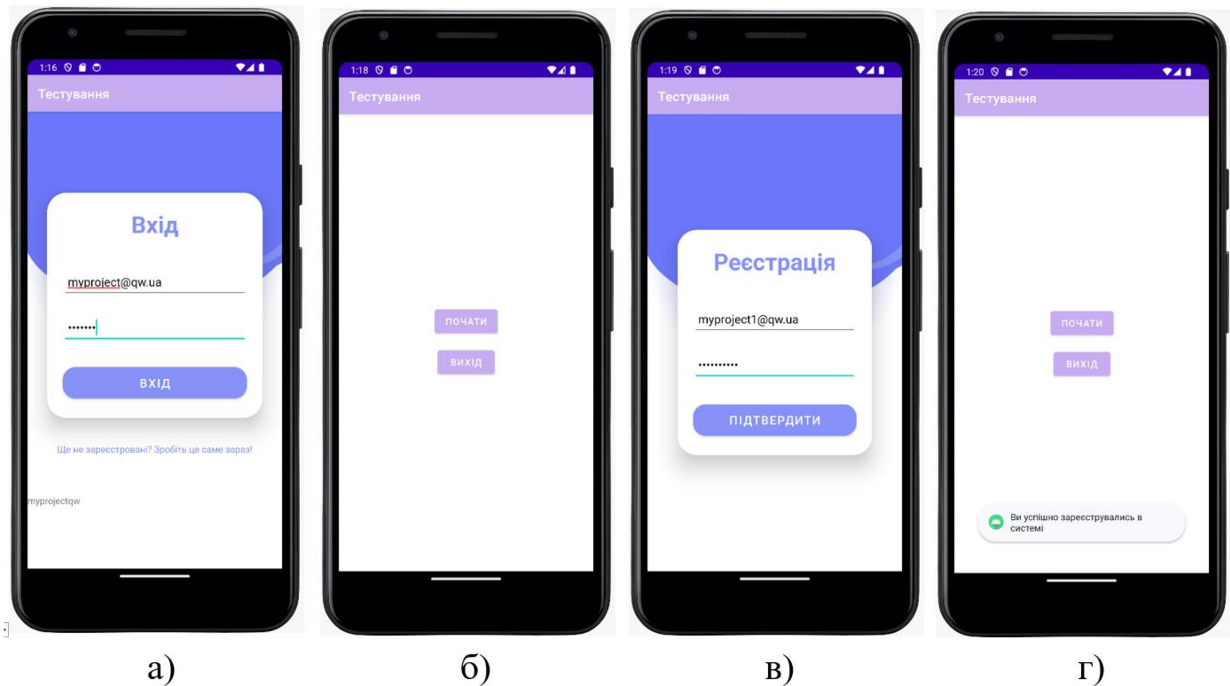
б)

Рисунок 3.15 – Лістинг коду «переходів»

На цьому розробку коду, що реалізує функціональну роботу самого тесту завершено.

3.3.2 Тестування розробленого коду

Проведемо перевірку на здійснення переходів між активностями LoginActivity – MainActivity (див. рис. 3.16, а, б) та RegistrationActivity – MainActivity (див. рис. 3.16, в, г).



а) б) в) г)

Рисунок 3.16 – Перевірка «переходів»

Логічні переходи між активностями виконуються коректно та без помилок (див. рис 3.16, б, г), клас MainActivity теж працює. Зробимо перевірку на працездатність всієї частини коду, яка відповідає за реалізацію процесу тестування. Активність MainActivity (див. рис. 3.17, а): при натисканні кнопки «Почати» здійснюється запуск активності TestActivity, а при натисканні кнопки «Вихід» здійснюється завершення роботи застосунку та повернення на «домашній екран» пристрою.

Активність TestActivity (див. рис. 3.17, б), при старті активності, одразу запускається таймер, який веде зворотній відлік від 60 секунд, та активуються кнопки «Так» і «Ні». При натискані цих кнопок, здійснюється обробка обраного варіанту та підрахунок правильних відповідей. Коли час

вичерпано, здійснюється перехід з пересиланням результату тесту до активності ResultActivity.

Активність ResultActivity (див. рис. 3.17, в) відображає на екрані отриманий результат тесту, та за допомогою кнопок «Спробувати ще» та «На головну» пропонує – пройти тест ще раз або повернутись до активності MainActivity.

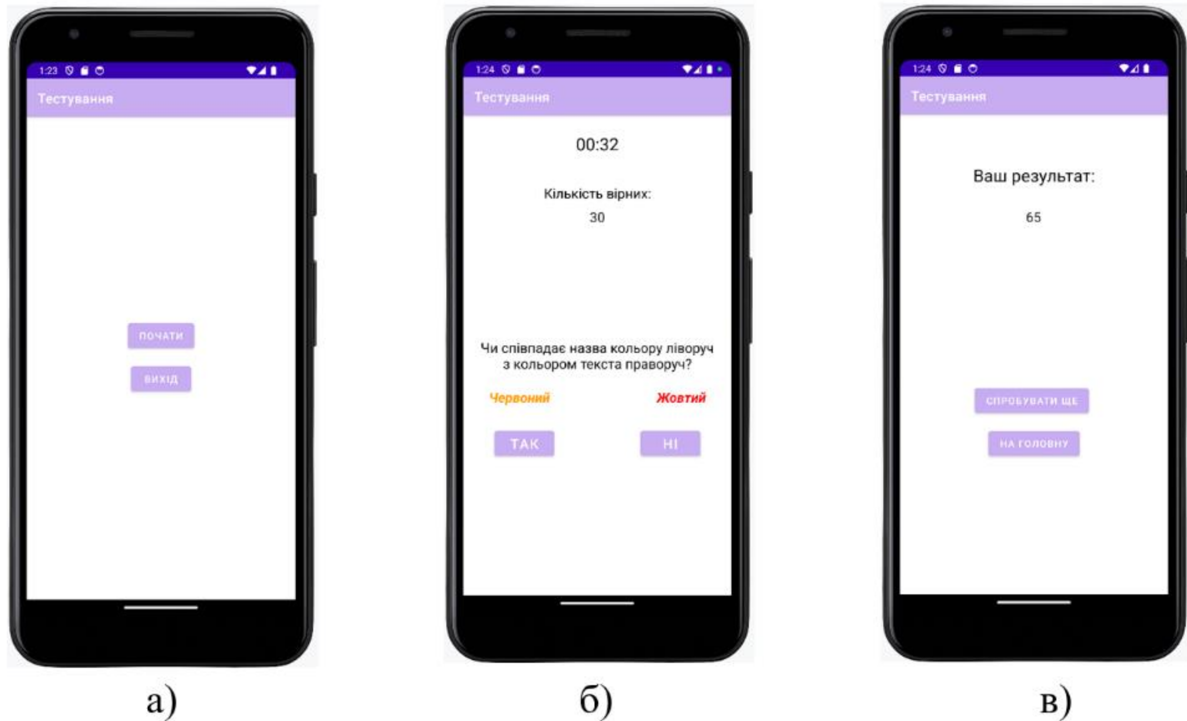


Рисунок 3.17 – Перевірка коду, що відповідає за тестування

Отже, можна вважати, що тестова перевірка коду, який відповідає за реалізацію процесу тестування пройдено вдало.

3.4 Інтеграція сервісу ML Kit FaceDetector

3.4.1 Розробка коду для реалізації сервісу від ML Kit FaceDetector

Для визначення «одноосібності» під час тестування використовується сервіс ML Kit FaceDetector. Для підключення цього сервісу імплементуємо до

складу проєкту відповідні бібліотеки (див. рис. 3.18).

```
// MlKit
implementation("com.google.android.gms:play-services-mlkit-face-detection:17.1.0")
implementation("com.google.mlkit:face-detection:16.1.5")
```

Рисунок 3.18 – Бібліотеки для роботи з ML Kit FaceDetector

При розробці цієї частини проєкту було створено 12 взаємозв'язаних файлів (чотири файли – class, три файли – data class, два файли – class: viewModel, два файли – з функціями, один файл – module), які виконують процес визначення облич, що знаходяться перед фронтальною камерою мобільного пристрою та передачі цієї інформації до блоку обробки. Розглянемо тільки деякі з них.

Клас FaceProcessor, який використано для обробки зображень з камери з метою виявлення обличчя за допомогою сервісу FaceDetector (див. рис. 3.19).

```
fun processImageProxy(image: ImageProxy, onDetectionFinished: (List<Face>) -> Unit) {
    detector.process(InputImage.fromMediaImage(image.image!!, image.imageInfo.rotationDegrees))
        .addOnSuccessListener(executor) { results: List<Face> -> onDetectionFinished(results) }
        .addOnFailureListener(executor) { e: Exception ->
            Log.e( tag: "Camera",  msg: "Error detecting face", e)
        }
        .addOnCompleteListener { image.close() }
}
```

Рисунок 3.19 – Обробка зображення для виявлення обличчя

Розглянемо процеси, що відбуваються в цьому методі. Він приймає ImageProxy (зображення з камери) та зворотній виклик onDetectionFinished, який викликається після завершення виявлення обличчя. Зображення конвертується в InputImage, яке може бути оброблене FaceDetector. Для цього використовується InputImage.fromMediaImage із врахуванням обертання зображення відносно камери (imageInfo.rotationDegrees). Метод detector.process використовується для запуску виявлення обличчя на

зображенні. Додавання обробників подій: addOnSuccessListener викликається при успішному виявленні обличчя і передає список облич (List<Face>) у зворотній виклик onDetectionFinished. Метод addOnFailureListener викликається при помилці виявлення обличчя і видає повідомлення про помилку до консолі лог. Метод addOnCompleteListener викликається завжди після успішного чи неуспішного виявлення, та відповідає за закриття ImageProxy, щоб звільнити ресурси.

Клас FaceDetectorViewModel, який розроблено за допомогою патерну архітектури MVVM для процесу запуску камери (див. рис. 3.20).

```
init {
    viewState.value = FaceDetectorViewState(
        camera = Camera(CameraSelector.LENS_FACING_FRONT,  isChanging: false)
    )
}

fun startCamera(activity: TestActivity, previewView: PreviewView) {
    cameraPreview.startCamera(
        activity = activity,
        previewView = previewView,
        cameraLens = viewState.requireValue().camera.lens,
        setSourceInfo = { it: ImageSourceInfo
            viewState.update { this: FaceDetectorViewState
                copy(imageSourceInfo = it)
            }
        },
        onFacesDetected = { it: List<Face>
            viewState.update { this: FaceDetectorViewState
                copy(detectedFaces = it)
            }
        }
    )
```

Рисунок 3.20 – Запуск камери через cameraPreview

В блоці ініціалізації встановлюється початковий стан FaceDetectorViewState, та створюється об'єкт Camera з параметрами: LENS_FACING_FRONT в статусі false (не змінюється).

Метод startCamera запускає камеру через cameraPreview, визначає параметри камери, включаючи activity, previewView та параметри зі стану viewState. Під час запуску камери визначається обробник інформації про джерело зображення setSourceInfo та для виявлення обличчя onFacesDetected.

Клас CameraPreview, який відповідає за підготовку та взаємодію з камерою в застосунку. В цьому класі знаходяться два важливі методи – startCamera (див. рис. 3.21) та buildImageAnalysis (див. рис. 3.22).

```
fun startCamera(
    activity: TestActivity,
    previewView: PreviewView,
    cameraLens: Int,
    setSourceInfo: (ImageSourceInfo) -> Unit,
    onFacesDetected: (List<Face>) -> Unit
) {
    val cameraProviderFuture = ProcessCameraProvider.getInstance(activity)

    cameraProviderFuture.addListener({
        val preview = Preview.Builder() Preview.Builder
            .build() Preview
            .apply { this: Preview
                setSurfaceProvider(previewView.surfaceProvider)
            }

        val cameraSelector: CameraSelector = CameraSelector.Builder().requireLensFacing(cameraLens).build()
        val analysis = buildImageAnalysis(cameraLens, setSourceInfo, onFacesDetected)
        val cameraProvider: ProcessCameraProvider = cameraProviderFuture.get()

        cameraProvider.unbindAll()
        cameraProvider.bindToLifecycle(activity, cameraSelector, preview, analysis)
    }, ContextCompat.getMainExecutor(activity))
}
```

Рисунок 3.21 – Ініціалізація та запуск камери

Метод startCamera ініціалізує та запускає камеру для попереднього перегляду в переданому PreviewView.

Він використовує ProcessCameraProvider для отримання постачальника камери, налаштовує попередній перегляд та аналіз, а також викликає метод buildImageAnalysis для налаштування аналізатора зображення.

Метод buildImageAnalysis створює об'єкт ImageAnalysis та налаштовує його аналізатор для обробки зображення камери.

Він викликає метод getSourceInfo для отримання інформації про джерело зображення та обробляє кожне отримане зображення за допомогою об'єкта faceProcessor.

Ці дві функції разом визначають основний цикл роботи з камерою та обробкою зображень обличчя в додатку.

```
private fun buildImageAnalysis(
    lens: Int,
    setSourceInfo: (ImageSourceInfo) -> Unit,
    onFacesDetected: (List<Face>) -> Unit
): ImageAnalysis {
    var sourceInfoUpdated = false

    return ImageAnalysis.Builder().build().apply { this: ImageAnalysis
        setAnalyzer(
            TaskExecutors.MAIN_THREAD
        ) { imageProxy: ImageProxy ->
            if (!sourceInfoUpdated) {
                setSourceInfo(getSourceInfo(lens, imageProxy))
                sourceInfoUpdated = true
            }

            faceProcessor.processImageProxy(imageProxy, onFacesDetected)
        }
    }
}
```

Рисунок 3.22 – Налаштування аналізатора та обробка зображень

Клас FaceView є підкласом View і використовується для відображення облич на екрані. Він має два основних поля: faces, яке зберігає список облич на зображенні, і imageSourceInfo, яке зберігає інформацію про джерело зображення (див. рис. 3.23).

Метод invalidateFaceView призначений для оновлення відображення обличчя. Він отримує список виявлених облич (detectedFaces) та інформацію про джерело нового зображення (newImageSourceInfo).

Після оновлення полів відбувається виклик методу invalidate, який призведе до оновлення відображення елемента у користувацькому інтерфейсі.

При кожному оновленні також викликається метод onNumberOfFacesChanged, який повідомляє слухача про зміни в кількості облич на зображенні.

В цьому файлі також додано інтерфейс, який визначає метод onNumberOfFacesChanged, який описано вище.

```
class FaceView(
    context: Context,
    private val listener: FaceViewListener
) : View(context) {

    private var faces: List<Face> = emptyList()
    private var imageSourceInfo: ImageSourceInfo = ImageSourceInfo(
        width: 10,
        height: 10,
        isImageFlipped: false)

    fun invalidateFaceView(detectedFaces: List<Face>,
                           newImageSourceInfo: ImageSourceInfo) {
        faces = detectedFaces
        imageSourceInfo = newImageSourceInfo
        val numOfFaces = detectedFaces.size
        listener.onNumberOfFacesChanged(numOfFaces)
        invalidate()
    }
}

interface FaceViewListener {
    fun onNumberOfFacesChanged(numberOfFaces: Int)
}
```

Рисунок 3.23 – Лістинг класу FaceView

Тепер зробимо деякі зміни безпосередньо в класі TestActivity, з якого будемо робити запуск сервісу FaceDetector та обробку отриманих від нього результатів.

Зробимо реалізацію інтерфейсу FaceViewListener (див. рис. 3.24), додаємо властивість previewView, яка використовує затриману ініціалізацію,

```
class TestActivity : AppCompatActivity(R.layout.activity_test), FaceViewListener {
    private val previewView by lazy { findViewById<PreviewView>(R.id.previewView) }
    private val viewModel: FaceDetectorViewModel by viewModel()
    private val faceView by lazy { FaceView( context: this,  listener: this) }
```

Рисунок 3.24 – Додавання та ініціалізація нових екземплярів та властивостей

щоб забезпечити інстанцію PreviewView під час першого звернення до цієї властивості, оголошуємо та ініціалізуємо екземпляр FaceDetectorViewModel за допомогою функції-розширення viewModel, який буде пов'язаний із

життєвим циклом активності, використовуємо затриману ініціалізацію для створення екземпляру FaceView, якому передається посилання на поточний контекст (this) та сам TestActivity в якості слухача (FaceViewListener).

Реалізуємо метод onNumberOfFacesChanged (див. рис. 3.25), який викликається при зміні кількості облич у детекторі обличчя. Метод отримує параметр numberOfFaces, який вказує на поточну кількість виявлених облич.

Перевіряється, чи кількість облич не перевищує 1, якщо кількість облич більше 1, відтворюється короткий звуковий сигнал (за допомогою MediaPlayer), щоб привернути увагу користувача до порушення умов тесту та виводиться коротке повідомлення за допомогою Toast, яке повідомляє користувача, що тест може проходити лише одна людина.

```
// імплементуємо метод інтерфейсу
override fun onNumberOfFacesChanged(numberOfFaces: Int) {
    Log.d( tag: "FaceDetector",  msg: "Кількість облич: $numberOfFaces")

    runOnUiThread {
        when {
            numberOfFaces > 1 -> {
                Toast.makeText(
                     context: this,
                     text: "Тест повинна проходити одна людина!",
                    Toast.LENGTH_SHORT).show()
                val mediaPlayer = MediaPlayer.create( context: this, R.raw.short_beep)
                mediaPlayer.start()
            }
        }
    }
}
```

Рисунок 3.25 – Лістинг методу onNumberOfFacesChanged

У методі onCreate (див. рис. 3.26) додаємо перевірку дозволу на використання камери за допомогою функції cameraPermissionGranted(this). Якщо дозвіл на використання камери надано, викликаємо функцію startCameraAndTimer, яка запускає камеру та таймер. Якщо дозвіл на використання камери не надано, викликаємо функцію requestPermissions(this), яка запитує користувача про надання дозволу на використання камери. Створюємо об'єкт спостереження observer, який

слідкує за змінами стану FaceDetectorViewState в viewModel.viewState. Викликаємо метод observe, який дозволяє об'єкту viewModel.viewState повідомляти про зміни, і при цьому викликати метод onStateChanged, оновлюється faceView за допомогою invalidateFaceView із новим станом облич та інформацією про джерело зображення. Також викликається метод onNumberOfFacesChanged, який отримує нову кількість облич і викликається відповідна обробка цієї інформації.

```
if (cameraPermissionGranted( activity: this)) {
    isCameraPermissionGranted = true
    startCameraAndTimer()
} else {
    requestPermissions( activity: this)
}

val observer = Observer<FaceDetectorViewState> { viewState ->
    onStateChanged(viewState)
    faceView.invalidateFaceView(viewState.detectedFaces, viewState.imageSourceInfo)
    onNumberOfFacesChanged(viewState.detectedFaces.size)
}

viewModel.viewState.observe( owner: this, observer)
```

Рисунок 3.26 – Перевірка дозволу та ініціалізація спостерігача

Метод startCamera, який запускає камеру та об'єкт CountDownTimer, що реалізує таймер, помістимо в метод startCameraAndTimer (див. рис. 3.27).

```
private fun startCameraAndTimer() {
    startCamera()

    object : CountDownTimer( millisInFuture: 60000,  countDownInterval: 1000) {
        @SuppressLint("SetTextI18n")
        override fun onTick(l: Long) {
            if (l < 10000) time!!.text = "00:0" + (l / 1000).toString()
            else time!!.text = "00:" + (l / 1000).toString()
```

Рисунок 3.27 – Вкладення таймера і старту камери до нового метода

Додаємо метод onRequestPermissionsResult (див. рис. 3.28), який викликається, коли користувач надає чи відмовляє в наданні дозволу після

виклику requestPermissions. Якщо користувач відмовив у наданні дозволу, виводиться повідомлення, що тест не буде доступний без дозволу на використання камери. Після цього активність TestActivity завершується методом onDestroy. Якщо дозвіл наданий, змінна isCameraPermissionGranted встановлюється в значення true, і викликається метод startCameraAndTimer, який запускає камеру та таймер.

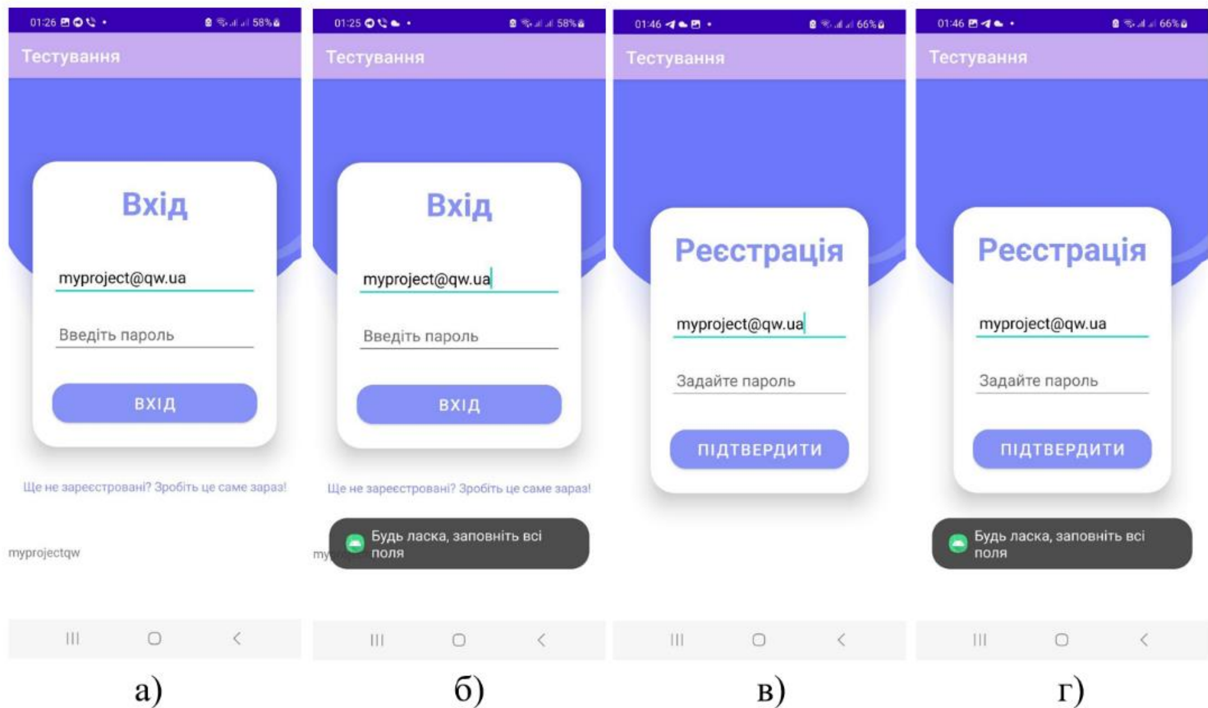
```
override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<String?>,
    grantResults: IntArray
) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == 0) {
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            isCameraPermissionGranted = true
            startCameraAndTimer()
        } else {
            Toast.makeText(
                context: this,
                text: "На жаль, без підключення камери, тест не доступний",
                Toast.LENGTH_LONG).show()
            onDestroy()
        }
    }
}
```

Рисунок 3.28 – Лістинг методу onRequestPermissionsResult

3.4.2 Тестування розробленого коду

У зв'язку з тим, що емулятор мобільного пристрою від IDE Android Studio не підтримує функціонування камери, проводити перевірку роботи застосунку будемо за допомогою мобільного телефону моделі Samsung SM-A715F API 33. Для цього під'єднаємо мобільний пристрій до комп'ютера за допомогою кабелю USB і після надання необхідних дозволів для USB підключення, зі списку Device Manager обираємо samsung SM-A715F. Спочатку зробимо ще декілька тестів з активностями LoginActivity (див.

рис. 3.29, а, б) та RegistrationActivity (див. рис. 3.29, в, г), а саме на спробу введення неповної інформації, а саме – не введено значення password.



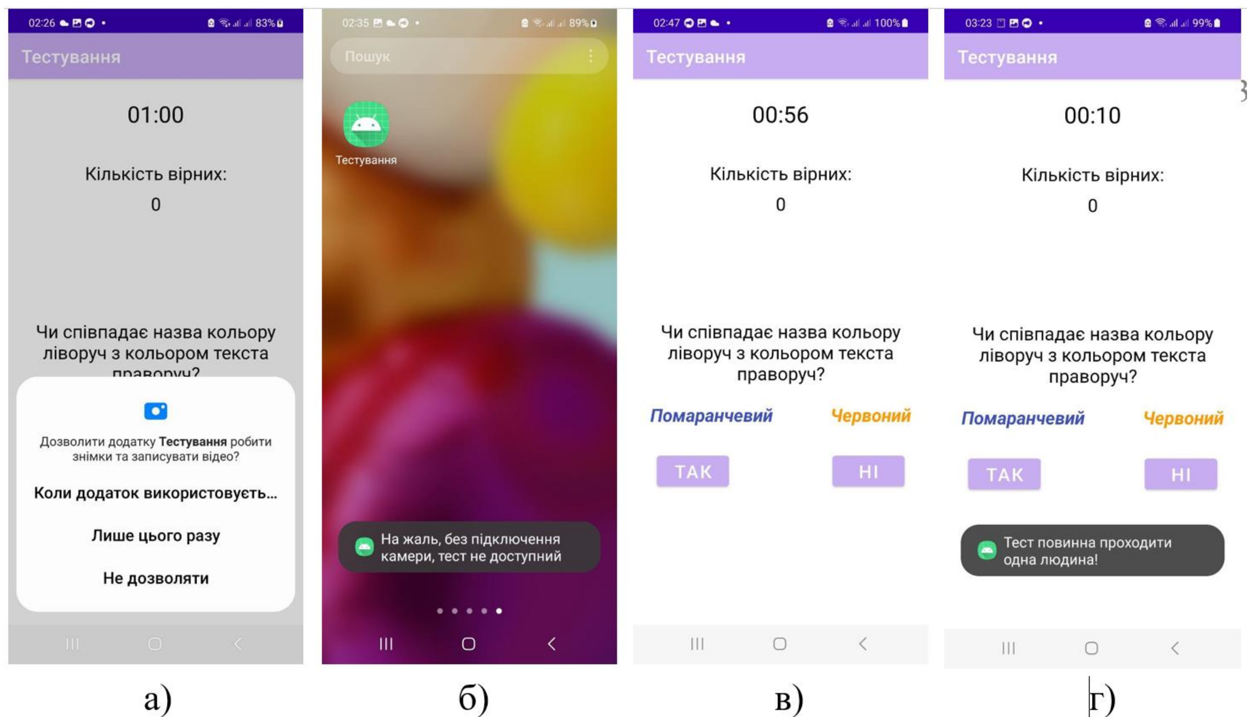
а) б) в) г)

Рисунок 3.29 – Тестування на введення неповної інформації

Перевіримо роботу сервісу FaceDetector. Після запуску активності TestActivity, так як для цього застосунку не було задано дозволів, то при обробці цієї події було відправлено запит до користувача (див. рис. 3.30, а), і поки не надано дозвіл на використання камери, таймер не працює. Спробуємо відмовити у дозволі на використання камери, отримали результат завершення роботи застосунку (див. рис. 3.30, б). Дозволимо використовувати камеру при роботі застосунку. Як тільки дозвіл отримано, одразу активується таймер, це значить що тестування почалось (див. рис. 3.30, в). Тест на визначення кількості облич, що потрапляють у об'єктив фронтальної камери показав наступні результати:

– поки перед камерою кількість облич не перевищує значення один, процес тестування проходить у звичайному режимі;
– якщо перед камерою кількість облич стає більшою за одне, метод onNumberOfFacesChanged відтворює звуковий сигнал та виводить

повідомлення, що тест повинна проходити одна людина (див. рис. 3.30, г).



а) б) в) г)

Рисунок 3.30 – Тестування перевірки дозволів та сервісу FaceDetector

Перевірка цієї частини коду проєкту Android застосунку також пройшла вдало. Таким чином можна зробити висновок, що застосунок який було розроблено для ОС Android на мові програмування Kotlin з інтеграцією сервісів Firebase, таких як Authentication та ML Kit FaceDetector пройшов тестові перевірки на всіх етапах розробки та повністю відповідає умовам завдання кваліфікаційної роботи.

ВИСНОВКИ

Результатом кваліфікаційної роботи є розроблений Android застосунок на мові програмування Kotlin представляє собою високотехнологічний інструмент, до складу якого інтегровані сервіси Firebase, такі як Authentication та ML Kit FaceDetector, з функцією тестування знань або навичок користувача.

Інтеграція Firebase Authentication гарантує безпеку та надійність процесу аутентифікації користувачів. Застосування ML Kit FaceDetector розширює можливості застосунку, дозволяючи визначати критерій «одноосібності» під час тестування, а також в перспективі, є можливість зробити обробку фіксації напрямку погляду для виявлення відволікання від процесу тестування.

Цей застосунок відкриває нові можливості для тестування та оцінювання знань чи навичок користувача в різних галузях. Функціональна гнучкість тестової частини дозволяє адаптувати застосунок до конкретних потреб відповідної галузі чи освітнього контексту.

Використання FaceDetector для визначення «одноосібності» та фіксації напрямку погляду додає нові рівні аналізу поведінки користувача під час тестування, що може покращити об'єктивність результатів та забезпечити додатковий рівень діагностики відволікань.

Отже, розроблений застосунок поєднує в собі передові технології та функціональність, що робить його ефективним інструментом для тестування та оцінювання, який може знаходити застосування в різних сферах, включаючи освіту, бізнес та персональний розвиток.

ПЕРЕЛІК ПОСИЛАНЬ

1. Ardito L., Coppola R., Malnati G., Torchiano M. Effectiveness of Kotlin vs Java in android app development tasks. *Information and Software Technology*. 2020. URL: https://doi.org/10.1016/j.infsof.2020.106374 (дата звернення: 05.08.2023).
2. Boduch A. React and React Native. Birmingham – Mumbai : Packt Publishing Ltd., 2017. 500 p.
3. Chougale P., Yadav V., Dr. Gaikwad A. Firebase – Overview and usage. *International Research Journal of Modernization in Engineering Technology and Science*. 2021. P. 1178–1183. URL: https://www.researchgate.net/profile/Anil-Gaikwad-12/publication/362539877_FIREBASE_-OVERVIEW_AND_USAGE/links/630ca17cacd814437fe6bca2/FIREBASE-OVERVIEW-AND-USAGE.pdf (дата звернення: 05.08.2023).
4. De Carli S. Authenticate Users with Firebase Auth. In: Build Mobile Apps with SwiftUI and Firebase. Apress, Berkeley, CA. 2023. P. 71–94. URL: https://doi.org/10.1007/978-1-4842-9452-9_4 (дата звернення: 12.09.2023).
5. Dimitrijevic N., Zdravkovic N., Milicevic V. An automated grading Framework for the mobile Development Programming Language Kotlin. *Faculty of Mechanical and Civil Engineering, Kraljevo,* 2023. URL: https://scidar.kg.ac.rs/bitstream/123456789/19059/1/AN%20AUTOMATED%20GRADING%20FRAMEWORK_Dimitrijevic_et_al.pdf (дата звернення: 18.09.2023).
6. Ebel N. Mastering Kotlin. Birmingham, Mumbai : Packt Publishing Ltd., 2019. 399 p.
7. Firebase Analytics. Google Developers. URL: https://web.archive.org/web/20160910080429/https://firebase.google.com/docs/analytics/ (дата звернення: 05.08.2023).
8. Firebase Auth. Firebase, Inc. URL:

https://web.archive.org/web/20160921013938/https://firebase.google.com/docs/auth/ (дата звернення: 12.09.2023).

9. Firebase Cloud Messaging. Google Developers. URL: https://web.archive.org/web/20160720185245/https://firebase.google.com/docs/cloud-messaging/ (дата звернення: 12.09.2023).

10. Firebase ML Kit Documentation. Google Developers. URL: https://firebase.google.com/docs/ml-kit (дата звернення: 12.09.2023).

11. Firebase Realtime Database. *Firebase, Inc.* URL: https://firebase.google.com/docs/database/ (дата звернення: 25.09.2023).

12. Firebase Storage. *Google Developers*. URL: https://firebase.google.com/docs/storage/ (дата звернення: 25.09.2023).

13. Horton J. Android Programming with Kotlin for Beginners. Birmingham, Mumbai : Packt Publishing Ltd., 2019. 665 p.

14. Jawad H. M. Android Mobile App Development as a Motivation towards Computer Programming. *IEEE International Conference on Electro Information Technology (EIT).* 2019. P. 169–175. URL: https://sci-hub.se/10.1109/EIT.2019.8833858 (дата звернення: 12.09.2023).

15. Karlsson J., Hindrikes D. Xamarin.Forms Projects. Birmingham, Mumbai : Packt Publishing Ltd., 2018. 391 p.

16. Kuprenko V. ML Kit for Firebase: features, capabilities, pros and cons. *Towards Data Science*. 2019. – URL: https://towardsdatascience.com/ml-kit-for-firebase-features-capabilities-pros-and-cons-a182b4299cc (дата звернення: 05.08.2023).

17. Laurence P.-O., Hinchman-Dominguez A. with Blake Meike G. & Dunn M. Programming Android with Kotlin. Sebastopol : Published by O’Reilly Media Incorporation, 2021. 339 p.

18. Martinez M., Mateus B.G. How and Why did developers migrate Android. Applications from Java to Kotlin? A study based on code analysis and interviews with developers. *Universit´e Polytechnique Hauts-de-France*, France. 2020. P. 1–29. URL: https://www.researchgate.net/profile/Matias-Martinez-

14/publication/340295591_How_and_Why_did_developers_migrate_Android_Applications_from_Java_to_Kotlin_A_study_based_on_code_analysis_and_interviews_with_developers/links/6151b724d2ebba7be74f1a8d/How-and-Why-did-developers-migrate-Android-Applications-from-Java-to-Kotlin-A-study-based-on-code-analysis-and-interviews-with-developers.pdf (дата звернення: 18.09.2023).

19. ML Kit for Firebase. Documentation. Google Developers. URL: https://firebase.google.com/docs/ml-kit (дата звернення: 12.09.2023).

20. ML Kit for Firebase. Machine learning for mobile developers. *Firebase, Inc.* URL: https://firebase.google.com/products/ml-kit/ (дата звернення: 05.08.2023).

21. Payne R. Developing in Flutter. Apress, Berkeley, CA, 2019. P. 9–27. URL: https://doi.org/10.1007/978-1-4842-5181-2_2 (дата звернення: 25.09.2023).

22. Smyth N. Jetpack Compose 1.4 Essentials. ebook | Payload Media, Incorporated. 2023. 608 p. URL: https://www.overdrive.com/media/10130306/jetpack-compose-1-4-essentials (дата звернення: 25.09.2023).