Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

**Бакалаврська кваліфікаційна робота**

**на тему:**

# «МЕРЕЖНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПЕРАТИВНОЇ КОМУНІКАЦІЇ НА ПЛАТФОРМІ ANDROID»

Виконав: студент 4 курсу , групи 2СП-20б
спеціальності 123 «Комп'ютерна інженерія»
освітня програма «Системне програмування»
_____Атаманюк Н.Р.
Керівник: к.т.н., проф. професор каф. ОТ
_____Захарченко С.М.
«10» 06 2024 р.
Рецензент: зав. каф. МБІС, к.т.н., доц.
_____Карпінець В.В.
«11» 06 2024 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.

«13» 06 2024 р.

Вінниця ВНТУ — 2024 рік

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженеріїКафедра обчислювальної техніки

Освітньо—кваліфікаційний рівень бакалавр

Галузь знань — 12 «Інформаційні технології»

Спеціальність — 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ОТ

проф., д.т.н. Азаров О. Д.

"06" лютого 2024 року

**З А В Д А Н Н Я**

**НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ**

Атаманюку Назару Романовичу

1 Тема роботи: «Мережне програмне забезпечення для оперативної комунікації на платформі Android», керівник роботи Захарченко С.М. к.т.н., проф. кафедри ОТ, затверджені наказом вищого навчального закладу від «11» березня 2024 року № 80

2 Термін подання студентом роботи 10.06.2024

3 Вихідні дані до роботи — огляд сучасних додатків для оперативної комунікації на платформі Android, аналіз методів розробки, розроблений додаток для оперативної комунікації на платформі Android.

4 Зміст розрахунково-пояснювальної записки: вступ, огляд предметної області, аналогів та методів розробки, налаштування проекту Android Studio та реалізація функціоналу додатку, тестування додатку, висновки, перелік посилань, додатки.

5 Перелік графічного матеріалу — структурна схема.

6 Консультанти розділів роботи представлені у таблиці 1.

Таблиця 1 — Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 – 3 | к.т.н., проф. каф. ОТ Захарченко С. М. | | |

7 Дата видачі завдання 12.03.2024 р.

8 Календарний план виконання БДР наведений у таблиці 2.

Таблиця 2 — Календарний план

| № | Назва та зміст етапу | Термін виконання | Примітка |
|---|---|---|---|
| 1. | Постановка задачі роботи | 1.03.24 | Викон. |
| 2. | Огляд сучасних додатків для оперативної комунікації на платформі Android | 02.03-25.03.24 | Викон. |
| 3. | Аналіз потреб користувачів | 26.03-05.04.24 | Викон. |
| 4. | Аналіз методів розробки | 06.04-10.04.24 | Викон. |
| 5. | Розробка додатку для оперативної комунікації на платформі Android | 11.04-30.04.24 | Викон. |
| 6. | Тестування додатку для оперативної комунікації на платформі Android | 01.05-06.05.24 | Викон |
| 7. | Оформлення пояснювальноїзаписки | 07.05-30.05.24 | Викон |
| 8. | Перевірка якості виконання бакалаврської роботи | 31.05-10.06.24 | Викон |

Студент _____ Атаманюк Н.Р.

Керівник роботи _____ Захарченко С.М.

# АНОТАЦІЯ

УДК 004.738.5

Атаманюк Н. Р. Мережне програмне забезпечення для оперативної комунікації на платформі Android. Бакалаврська кваліфікаційна робота зі спеціальності 123 «Комп'ютерна інженерія», освітня програма «Системне програмування». Вінниця: ВНТУ, 2024. 110 с.

На укр. Мові. Бібліогр.; 36 рис.; 39 лістингів; табл. 2.

У бакалаврській кваліфікаційній роботі розроблено мережне програмне забезпечення для оперативної комунікації на платформі Android. Додаток дозволяє покращити ефективність обміну інформацією серед користувачів мобільних пристроїв. У загальній частині роботи розглянуто особливості побудови сучасних мобільних комунікаційних систем та обґрунтовано доцільність їх розробки.

Проект реалізовано з використанням Android Studio та мовою програмування Java. Для забезпечення функціональності додатку застосовано сервіси Firebase, зокрема Authentication, Realtime Database та Storage. Створено наступні активності: splash screen, login, registration, main activity, chat window та setting. У конструкторській частині виконано розробку архітектури додатку. У технологічній частині проведено налаштування системи, представлено методику інтеграції Firebase сервісів та налагодження додатку.

Ключові слова: Android Studio, Firebase, активність, обмін повідомленнями, Realtime Database, Java.

# ABSTRACT

Atamaniuk N. R. Network software for operational communication on the Android platform. Bachelor's thesis on specialty 123 "Computer engineering", educational program "System programming". Vinnytsia: VNTU, 2024. 110 pages.

In Ukrainian. Bibliography; 36 figures; 39 listings; 2 table.

In the bachelor's thesis, network software for operational communication on the Android platform has been developed. The application aims to enhance the efficiency of information exchange among mobile device users. The general part of the work examines the features of modern mobile communication systems and justifies the feasibility of their development.

The project is implemented using Android Studio and the Java programming language. To ensure the functionality of the application, Firebase services such as Authentication, Realtime Database, and Storage have been used. The following activities were created: splash screen, login, registration, main activity, chat window, and setting. In the design part, the architecture of the application was developed. In the technological part, the system was configured, the methodology for integrating Firebase services, and the debugging of the application were presented.

Keywords: Android Studio, Firebase, activity, messaging, Realtime Database, Java.

**ЗМІСТ**

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

UI — (User Interface) Користувацький інтерфейс

SDK — (Software Development Kit) Набір із засобів розробки, утиліт і документації

VR — (Virtual Reality) Різновид реальності в формі тотожності матеріального й ідеального

AR — (Augmented Reality) Термін, що позначає всі проекти, спрямовані на доповнення реальності будь-якими віртуальними елементами

API — (Application Programming Interface) Набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом) для використання в зовнішніх програмних продуктах

UID — (User ID або user identifier) Ідентифікатор користувача

JSON — (JavaScript Object Notation) Текстовий формат обміну даними

URL — (Uniform Resource Locator) Стандартизована адреса певного ресурсу

# ВСТУП

У сучасному світі, де мобільні технології стали невід'ємною частиною повсякденного життя, оперативна комунікація відіграє ключову роль у забезпеченні ефективної взаємодії між людьми. Зростаюча популярність мобільних платформ, зокрема Android, створює необхідність у розробці високоякісного мережного програмного забезпечення, яке б задовольняло потреби користувачів у швидкому та надійному обміні інформацією. Створення такого програмного забезпечення є важливим як для особистого, так і для професійного використання, що робить дослідження у цій галузі надзвичайно актуальним.

В останні роки значна кількість досліджень була присвячена розробці та оптимізації мобільних додатків для комунікації. Відомі мобільні додатки, такі як WhatsApp, Telegram, Signal, демонструють високі стандарти у сфері мережного програмного забезпечення, проте існує потреба у подальших дослідженнях з метою вдосконалення цих технологій та адаптації їх до нових вимог користувачів. **Актуальність** даного дослідження підкреслюється необхідністю забезпечення швидкої та надійної комунікації на платформі Android.

**Метою** даного дослідження є розробка мережного програмного забезпечення для оперативної комунікації на платформі Android, з акцентом на простоту використання. У рамках цієї роботи проведено аналіз існуючих рішень та визначено їхні переваги, розроблено архітектуру програмного забезпечення для швидкої та надійної комунікації, реалізовано функціональні можливості додатку, зокрема обмін текстовими повідомленнями, а також проведено тестування додатку для забезпечення високої якості та зручності користування.

**Об'єктом** дослідження є процеси оперативної комунікації у мобільних мережах на платформі Android.

**Предметом** дослідження є методи та засоби розробки мережного програмного забезпечення для оперативної комунікації на платформі Android. У

ході дослідження використовувались методи системного аналізу, проектування програмного забезпечення та тестування.

**Новизна** даного дослідження полягає у розробці та впровадженні мережного програмного забезпечення для оперативної комунікації, яке реалізоване з використанням платформи Firebase. Це дозволило створити додаток з високою швидкістю обміну повідомленнями та надійністю, що є важливим для сучасних користувачів.

Результати дослідження пройшли **апробацію** шляхом публікації додатку на Amazon Appstore, що дозволило отримати цінний досвід щодо його використання в реальних умовах.

Також результати дослідження були **опубліковані** на міжнародній науково-методичній Інтернет-конференції «Проблеми вищої математичної освіти: виклики сучасності» в секції «теоретико-методологічні та психологічні аспекти створення і впровадження інформаційно-комунікаційних та інноваційних технологій» з назвою «Використання Firebase для розробки мережного програмного забезпечення для оперативної комунікації на платформі Android».

# 1 ОГЛЯД АНАЛОГІВ ТА МЕТОДІВ РОЗРОБКИ

## 1.1 Оперативна комунікація на мобільних платформах

Оперативна комунікація на мобільних платформах не просто стала невід'ємною складовою сучасного інформаційного ландшафту — вона перетворилася на ключовий елемент життя сучасної людини. З розвитком технологій та поширенням смартфонів вона стала доступною для широкого кола користувачів у всьому світі, забезпечуючи ними можливість швидкої та зручної спільноти незалежно від місця та часу.

Така форма комунікації включає в себе різноманітні інструменти, від текстових повідомлень до голосових та відео викликів, а також спільну роботу над документами. Месенджери та соціальні мережі, які спочатку були просто засобами спілкування, тепер перетворилися на справжні платформи для співпраці, розваг та організації подій. Вони об'єднують у собі не лише можливості обміну повідомленнями, але і спільну роботу над проектами, обмін файлами та організацію подій, створюючи віртуальний простір для всебічної взаємодії.

Зокрема, тренд розвитку оперативної комунікації на мобільних платформах полягає в інтеграції з іншими технологіями, такими як віртуальна реальність та розширена реальність [1]. Це відкриває нові можливості для взаємодії та спілкування, наприклад, можливість віртуальних нарад або відтворення сценаріїв спілкування у віртуальному середовищі.

Зростає значення персоналізації та адаптації комунікаційних платформ до індивідуальних потреб користувачів. Алгоритми штучного інтелекту дозволяють аналізувати поведінку користувачів та надавати рекомендації щодо оптимальних способів спілкування та взаємодії. Наприклад, за допомогою аналізу даних можна створювати персоналізовані пропозиції та повідомлення для клієнтів бізнесу або надавати індивідуально налаштовані інтерфейси користувачам для зручного спілкування та роботи з додатками.

У сфері бізнесу оперативна комунікація на мобільних платформах виявляється особливо важливою. Вона допомагає забезпечити ефективну

взаємодію між співробітниками, клієнтами та партнерами, дозволяючи швидко обмінюватися інформацією, реагувати на зміни та розв'язувати проблеми в режимі реального часу [2]. Наприклад, мобільні додатки для клієнтського обслуговування дозволяють користувачам швидко звертатися за підтримкою та отримувати відповіді на свої запити.

Проте разом зі зростанням значення оперативної комунікації зростає і важливість забезпечення конфіденційності та безпеки. Використання шифрування даних, захищених з'єднань та сучасних методів аутентифікації, дозволяє зберігати особисту інформацію в безпеці від несанкціонованого доступу та кіберзлочинців [3]. Постійне вдосконалення технологій та регулярне оновлення програмного забезпечення є ключовими для забезпечення високого рівня безпеки в цій області.

Оперативна комунікація на мобільних платформах може бути інтегрована з іншими сервісами та додатками, що дозволяє забезпечувати більш широкий спектр можливостей для користувачів. Наприклад, можливість надсилати миттєві повідомлення з картами, документами або мультимедійним вмістом робить комунікацію більш ефективною та продуктивною. Додатковою важливою характеристикою є мобільність, яка дозволяє користувачам комунікувати з будь-якого місця та в будь-який час, використовуючи лише свій смартфон чи планшет [4]. Це робить оперативну комунікацію на мобільних платформах особливо зручною для людей, які постійно перебувають в русі або працюють з віддаленими командами.

Крім того, розвиток квантових технологій може вплинути на безпеку та конфіденційність комунікаційних систем. Квантова криптографія може забезпечити надійний захист від кібератак, оскільки вона базується на принципах квантової механіки, які гарантують неможливість перехоплення інформації без виявлення змін у квантовому стані системи.

Майбутнє оперативної комунікації на мобільних платформах обіцяє ще більше інновацій та розширення можливостей для користувачів. Розвиток штучного інтелекту відкриває нові можливості для автоматизації процесів

комунікації та підвищення їх ефективності. Подальше вдосконалення методів шифрування та захисту даних стане ключовим завданням для забезпечення приватності та безпеки користувачів у майбутньому.

## 1.2 Огляд та аналіз існуючих рішень для оперативної комунікації

Огляд та аналіз існуючих рішень для оперативної комунікації на мобільних платформах відображає різноманіття програмних продуктів, сервісів та технологій, що пропонують засоби взаємодії користувачів у реальному часі через мобільні пристрої. Цей сегмент ринку є вкрай конкурентним, так як він постійно розвивається, адаптуючись до потреб користувачів та новітніх технологій.

У світі месенджерів основними гравцями є програми, такі як WhatsApp, Telegram, Facebook Messenger, Viber, Signal, Line та інші. Кожен з них має свої унікальні особливості та функції, але всі вони спрямовані на забезпечення швидкої та зручної комунікації. Наприклад, WhatsApp має широке розповсюдження та широкий функціонал, включаючи відео виклики та групові чати, тоді як Telegram відомий своїм фокусом на приватність та безпеку, а також розширеними можливостями групової спільноти. Порівняння месенджерів описано в таблиці 1.1.

У сфері соціальних мереж домінують платформи, такі як Facebook, Twitter, Instagram, LinkedIn та Snapchat. Ці соціальні мережі надають не лише можливості спілкування, але і інструменти для створення контенту, спільної роботи над проектами, рекламних кампаній та багато іншого. Наприклад, Instagram славиться своїми функціями візуального спілкування через фотографії та відео, тоді як LinkedIn використовується для професійних зв`язків та пошуку роботи.

У сегменті бізнес-комунікації важливими інструментами є Slack, Microsoft Teams, Skype for Business та Google Meet. Ці платформи спеціалізуються на організації комунікації внутрішньої команди, спільної роботи над проектами та проведенні віддалених зустрічей та конференцій. Наприклад, Slack відомий своєю гнучкістю та інтеграцією з іншими інструментами, тоді як Microsoft Teams

відділяється своєю глибокою інтеграцією з іншими продуктами Microsoft та ефективною організацією робочих просторів.

Таблиця 1.1 — Порівняння месенджерів

| | **WhatsApp** | **Telegram** | **Facebook Messenger** | **Viber** | **Signal** | **Line** |
|---|---|---|---|---|---|---|
| **Власник** | Meta | Telegram Messenger LLP | Meta | Rakuten | Signal Foundation | Line Corporation |
| **Рік запуску** | 2009 | 2013 | 2011 | 2010 | 2014 | 2011 |
| **Платформи** | iOS, Android, Windows, macOS, Web | iOS, Android, Windows, macOS, Web | iOS, Android, Windows, macOS, Web | iOS, Android, Windows, macOS, Linux, Web | iOS, Android, Windows, macOS, Linux | iOS, Android, Windows, macOS, Web |
| **Групові чати** | До 256 учасників | До 200,000 учасників | До 250 учасників | До 250 учасників | До 1000 учасників | До 500 учасників |
| **Відео-дзвінки** | До 8 учасників | До 1000 учасників (тільки аудіо) | До 50 учасників | До 20 учасників | До 40 учасників | До 200 учасників |
| **Хмарне збереження повідомлень** | Так (Google Drive, iCloud) | Так (Telegram Cloud) | Так (Facebook сервера) | Так (Viber Backup) | Ні | Так (Line Cloud) |
| **Боти** | Ні | Так | Так | Так | Ні | Так |
| **Файли, що передаються** | До 100 MB | До 2 GB | До 25 MB | До 200 MB | До 100 MB | До 1 GB |
| **Доступність API** | Ні | Так | Так | Так | Ні | Так |

У зв'язку з різноманітністю вимог користувачів, додатки для комунікації постійно вдосконалюються та доповнюються новими функціями. Наприклад, на сучасних платформах часто з'являються можливості відео дзвінків, голосових

повідомлень, розсилання геолокації тощо, щоб забезпечити більш різноманітні можливості спілкування. Для забезпечення максимальної доступності, додатки для комунікації зазвичай розробляються для різних платформ, таких як iOS, Android, та навіть веб-версії [5]. Це дозволяє користувачам спілкуватися між собою, незалежно від того, який пристрій вони використовують. Якість зв'язку та стабільність роботи також відіграють велику роль. Користувачі очікують, що їхні повідомлення будуть доставлені миттєво, а відео дзвінки пройдуть без перебоїв. Тому розробники активно працюють над оптимізацією додатків, щоб забезпечити найвищу якість зв'язку та стабільну роботу навіть при поганих мережевих умовах.

Зрештою, кожна з цих платформ має свої переваги та недоліки, і вибір конкретного рішення залежить від потреб користувачів, їхніх уподобань та специфіки завдань, які вони хочуть вирішити через комунікацію на мобільних платформах.

## 1.3 Вибір мобільної платформи

Методологія та засоби розробки є важливими аспектами при створенні мобільних додатків. Особливу увагу приділяють вибору мобільної платформи, оскільки це визначає технічні аспекти розробки, а також аудиторію, яку буде охоплювати додаток. Методологія розробки визначає підхід та процес, який використовується під час створення мобільного додатка. Наприклад, Agile та Scrum  популярні методології, які надають гнучкість і можливість швидко адаптуватися до змін у вимогах проекту. Waterfall ідеально підходить для проектів зі стабільними вимогами, де розробка відбувається послідовно, від аналізу до реалізації.

iOS і Android є двома найпопулярнішими мобільними платформами, і вибір між ними може залежати від географічного розподілу аудиторії та її платіжної здатності [6]. Для більшої частини світу Android є більш популярною платформою через більш широкий асортимент пристроїв та більш доступні ціни, тоді як iOS зазвичай використовується в країнах з вищим рівнем доходів.

При виборі мобільної платформи розробники зазвичай розглядають такі фактори, як розповсюдженість платформи серед цільової аудиторії, її технічні можливості, екосистема розробки, доступність інструментів та ресурсів для створення додатків. Наприклад, iOS-користувачі частіше схильні до витрат на додатки, тоді як Android-користувачі можуть бути більш чутливими до цін та шукати альтернативи. Також важливо враховувати технічні особливості кожної платформи, такі як розміри екрану, підтримка різних пристроїв, особливості API та доступні можливості. Порівняння платформ IOS та Android описано у таблиці 1.2.

У разі вибору платформи розробки можуть використовуватися різні засоби та середовища. Наприклад, для розробки додатків на iOS використовується мова програмування Swift та середовище розробки Xcode, в той час як для Android - Java або Kotlin та середовище розробки Android Studio. Крім того, існують мультиплатформні фреймворки, такі як React Native або Flutter, які дозволяють розробникам створювати додатки для обох платформ з одного коду.

Зважаючи на швидкі технологічні зміни в мобільній сфері, розробники також повинні враховувати інші аспекти при створенні мобільних додатків. У зв'язку зі зростанням кількості кіберзлочинності, важливо забезпечити високий рівень захисту для особистих даних користувачів у мобільних додатках. Розробники повинні дотримуватися найкращих практик щодо шифрування даних, захисту від SQL-ін'єкцій та інших потенційних атак.

Кожна країна має власні правила і закони, які стосуються збору, зберігання та обробки даних користувачів. Розробники повинні ретельно вивчати ці законодавчі вимоги і забезпечити відповідність своїх додатків.

При розробці мобільних додатків важливо оптимізувати їх продуктивність для різних пристроїв і версій операційної системи. Це може включати оптимізацію швидкості завантаження, роботу в умовах обмеженої мережі та ефективне використання ресурсів пристрою, таких як об’єм заряду батареї та обсяг пам'яті.

Таблиця 1.2 — Порівняння платформ IOS та Android

| | **iOS** | **Android** |
|---|---|---|
| **Виробник** | Apple | Google |
| **Операційна система** | Закрита, лише для пристроїв Apple | Відкритий код, використовується на багатьох пристроях різних виробників |
| **Інтерфейс користувача** | Однорідний, контрольований Apple | Варіативний, залежить від виробника та налаштувань користувача |
| **Магазин додатків** | App Store | Google Play Store, а також інші магазини додатків |
| **Безпека** | Високий рівень безпеки, регулярні оновлення | Залежить від виробника |
| **Оновлення операційної системи** | Регулярні оновлення для всіх підтримуваних пристроїв одночасно | Залежить від виробника та оператора зв'язку, часто із затримкою |
| **Сумісність з пристроями** | Тільки Apple пристрої (iPhone, iPad, iPod) | Велика кількість пристроїв від різних виробників |
| **Кастомізація** | Обмежена, базовий рівень налаштувань | Високий рівень кастомізації |
| **Вбудовані сервіси** | Інтеграція з екосистемою Apple | Інтеграція з Google сервісами |
| **Голосові асистенти** | Siri | Google Assistant |
| **Ринок додатків** | Жорсткий контроль якості, обмеження на деякі типи додатків | Менше обмежень, більше різноманітності, але більше ризиків шкідливих додатків |
| **Підтримка** | Офіційна підтримка через Apple Store | Підтримка залежить від виробника пристрою |
| **Ціна** | Висока вартість пристроїв Apple | Широкий діапазон цін, від бюджетних до преміум моделей |
| **Адаптація до ринку** | Обмежений вибір моделей | Великий вибір моделей |
| **Спільнота розробників** | Суворі вимоги, висока вартість входу | Відкритий код, легше розпочати розробку |

Важливо включити механізми аналізу та збору зворотного зв'язку в мобільний додаток, щоб зрозуміти, як користувачі взаємодіють з додатком і як його можна покращити в майбутньому. Це може бути здійснено через вбудовані аналітичні інструменти або зовнішні сервіси аналізу даних.

Після випуску додатку важливо забезпечити його постійну підтримку та обслуговування, включаючи виправлення помилок, оновлення для сумісності з новими версіями операційної системи та вдосконалення функціональності на основі зворотного зв'язку від користувачів.

Ці аспекти разом з попередньо розглянутими методологіями та засобами розробки допомагають розробникам створювати мобільні додатки, які не тільки задовольняють потреби користувачів, але й забезпечують високий рівень якості, безпеки та продуктивності.

## 1.4 Обґрунтування вибору Android як операційної системи

Зважаючи на значущість операційної системи для розробки мобільного додатка, вибір Android є стратегічно обґрунтованим. Android як операційна система для розробки мобільного додатка може базуватися на ряді чинників:

— має найбільшу частку ринку серед операційних систем для мобільних пристроїв, що означає, що велика кількість користувачів вже використовує цю платформу, що дозволить вам охопити широку аудиторію, яка може бути особливо привабливою, якщо ваша мета — глобальне охоплення [7];

— відомий своєю гнучкістю, доступністю та працює на широкому спектрі пристроїв, від бюджетних до преміум-сегменту, що дозволяє користувачам з різними фінансовими можливостями отримати доступ до вашого додатка;

— надає розробникам широкий набір функціональних можливостей та API, що дозволяє створювати різноманітні додатки з різноманітними функціями, що означає, що ви маєте більше можливостей для створення унікальних та інноваційних продуктів;

— дозволяє користувачам кастомізувати свої пристрої та налаштовувати їх згідно з власними уподобаннями, що може бути важливим фактором для користувачів, і розробка для Android дозволяє створювати додатки, які враховують цю потребу;

— тісно інтегрований з іншими сервісами Google, такими як Google Maps, Google Drive, Gmail та багато інших, які відкривають додаткові можливості для розробки додатків, які використовують ці сервіси для поліпшення функціональності та досвіду користувача, наприклад інтеграція з Google Maps дозволяє створювати додатки з геолокаційними функціями, такими як навігація, місцевий пошук та інші сервіси, що покращують взаємодію з користувачем та роблять додаток більш привабливим.

Глибока інтеграція з Gmail або Google Drive може забезпечити безперервний обмін файлами або спілкування по електронній пошті в межах додатка, покращуючи продуктивність та зручність користувача [8]. Ця інтеграція не лише поліпшує функціональність додатка, але й підвищує його видимість та доступність в межах широкої екосистеми сервісів Google.

З урахуванням цих чинників вибір Android як операційної системи може бути раціональним рішенням, особливо якщо є прагнення до максимального охоплення аудиторії та широких можливостей розвитку.

## 1.5 Використання Android Studio для розробки додатків

Android Studio — це інтегроване середовище розробки програмного забезпечення (IDE), яке спеціально призначене для створення мобільних додатків для операційної системи Android. Його можливості включають в себе широкий набір інструментів та функціонал, спрямований на полегшення робочого процесу розробки та забезпечення високої продуктивності [9].

Однією з ключових переваг Android Studio є його інтеграція з Android SDK, що надає розробникам доступ до усіх необхідних ресурсів, бібліотек та інструментів для створення додатків. У склад Android Studio входить зручний редактор коду з розширеними функціями, такими як автодоповнення та

підсвічування синтаксису, а також інструменти для візуального дизайну інтерфейсу користувача, що дозволяють створювати привабливі та функціональні UI [10]. Крім того, Android Studio надає розробникам можливість тестувати свої додатки за допомогою вбудованих емуляторів Android, або підключати реальні пристрої для тестування на різних конфігураціях та версіях Android. Це дозволяє ефективно перевіряти та налагоджувати додатки перед їхнім випуском на ринок.

Додатково, Android Studio має глибоку інтеграцію з різними сервісами Google, такими як Google Maps, Firebase, Google Analytics та інші. Це дозволяє розробникам легко інтегрувати різноманітні функціональності та сервіси в свої додатки, що розширює можливості розробки та поліпшує користувацький досвід.

Android Studio підтримує різні мови програмування для розробки додатків, включаючи Java, Kotlin та C++. Kotlin, як офіційна мова програмування для Android, надає розробникам більше можливостей та зручності у порівнянні з традиційним Java.

Розширені можливості Android Studio включають в себе:

— підтримує інтеграцію з різними системами контролю версій, такими як Git, що дозволяє розробникам зручно керувати версіями свого коду та спільно працювати над проектами;

— має інтеграцію з різними фреймворками для розробки ігор, такими як Unity або Unreal Engine, що робить його відмінним вибором для розробки як мобільних додатків, так і ігор для платформи Android;

— має вбудований монітор профілювання, який дозволяє розробникам виявляти та виправляти проблеми з продуктивністю та використанням ресурсів в їхніх додатках;

— дозволяє розробникам створювати додатки, які оптимізовані для різних типів сучасних пристроїв, включаючи смартфони, планшети, Android TV та Android Wear;

— підтримує встановлення різноманітних плагінів, що дозволяє розширювати його функціональність та додавати нові можливості згідно з потребами розробника.

Загалом, Android Studio — це незамінний інструмент для розробників мобільних додатків, який надає широкі можливості та інструменти для створення високоякісних та функціональних додатків для платформи Android. Його зручний інтерфейс та розширена функціональність роблять його першим вибором для багатьох розробників по всьому світу.

### 1.6 Використання Firebase як серверної частини

Firebase - це платформа для розробки мобільних та веб-додатків, яка надає широкий спектр інструментів та сервісів для реалізації різноманітних функцій. Використання Firebase як серверної частини для мобільних додатків може стати важливим елементом розробки, оскільки його можливості полегшують роботу, підвищують якість та продуктивність додатків [11].

Для управління додатками Firebase надає Firebase Console, інтуїтивно зрозумілий інтерфейс, де розробники можуть аналізувати дані, налагоджувати додатки та відстежувати продуктивність. Функції Remote Config дозволяють змінювати вигляд та поведінку додатка без необхідності оновлення, а також проводити експерименти та тестування. За допомогою Firebase ML можна інтегрувати машинне навчання в додатки, використовуючи як попередньо навчені моделі, так і власні.

Firebase надає розширену аналітику для відстеження та аналізу використання додатків. Це включає в себе дані про користувачів, активність, конверсії, витрати та багато іншого, що дозволяє розробникам отримувати цінну інформацію для оптимізації своїх додатків. Платформа має можливості для інтеграції з рекламними мережами, такими як AdMob, що дозволяє розробникам монетизувати свої додатки через рекламу. Аналіз помилок та відстеження витрат Firebase Crashlytics надає інструменти для відстеження та аналізу помилок у вашому додатку, дозволяючи швидко виявляти та виправляти проблеми. Firebase Performance Monitoring дозволяє вимірювати час відгуку додатка, швидкість завантаження та інші метрики продуктивності.

Структурна схема того, яким чином можна працювати з платформою Firebase для розробки мобільного додатку, зображена на рисунку 1.1.

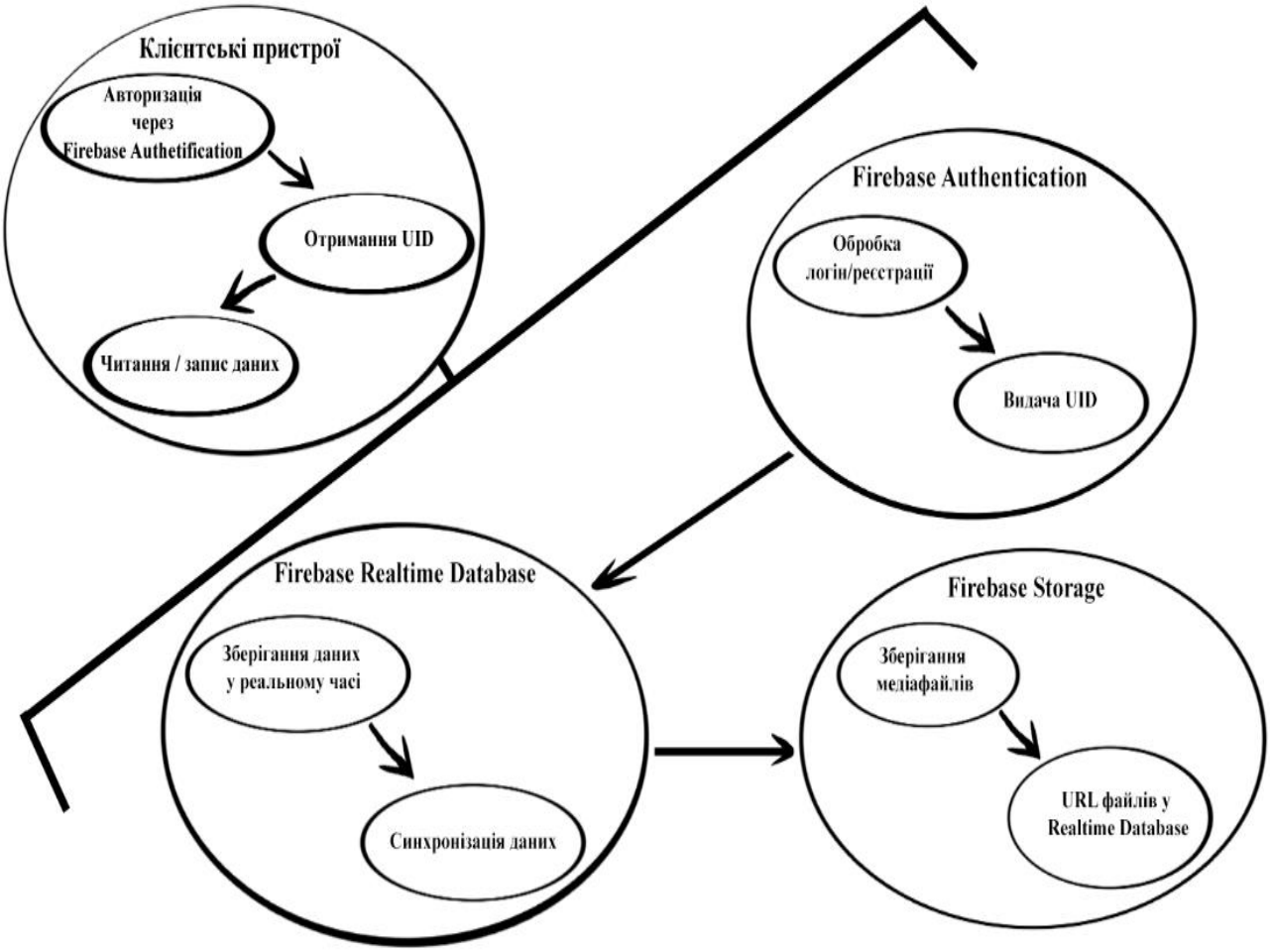


Рисунок 1.1 — Структурна схема роботи з платформою Firebase

За допомогою клієнтських пристроїв в цьому випадку є можливість:

— використовувати Firebase Authentication для авторизації користувачів, після чого користувач отримує унікальний ідентифікатор (UID);

— читати/записувати дані, тобто мобільний додаток взаємодіє з Firebase Realtime Database для зберігання та отримання повідомлень у реальному часі.

Firebase Authentication у цьому випадку:

— обробляє логін/реєстрацію користувачів через різні методи (електронна пошта/пароль, соціальні мережі, тощо);

— після успішної авторизації, кожному користувачу надає унікальний ідентифікатор (UID), який використовується для доступу до даних у Realtime Database і Storage.

Firebase Realtime Database використовується:

— для зберігання даних у вигляді JSON-структури;

— синхронізації даних у реальному часі, тобто будь-які зміни у базі даних негайно передаються всім підключеним клієнтам.

Firebase Storage:

— використовується для зберігання різних файлів, таких як зображення профілю, яке можна використовувати за допомогою посилання;

— зберігає файли, тобто URL-адреси файлів у Realtime Database для доступу з мобільного додатку.

Варто зазначити, що на безперервну підтримку та оновлення для Firebase впливають кілька факторів. По-перше, це активна розробка та вдосконалення платформи з боку команди Google. Вони регулярно випускають нові версії Firebase з оновленнями, новими функціями та поліпшеннями, щоб відповідати новим потребам розробників та ринку мобільних додатків [12].

По-друге, Firebase активно взаємодіє зі своєю спільнотою розробників. Вони збирають відгуки, пропозиції та запити на функціонал від користувачів та розробників, щоб зрозуміти, які інструменти та функції були б корисними. Цей діалог допомагає визначити пріоритети в розробці та спрямовує зусилля на те, що є найбільш важливим для спільноти [13].

Крім того, Firebase активно слідкує за останніми тенденціями у мобільній розробці та технологічних інноваціях. Вони швидко реагують на нові вимоги та можливості, що дозволяє їм постійно підтримувати високий рівень конкурентоспроможності Firebase на ринку мобільних розробок.

Загалом, використання Firebase як серверної частини дозволяє розробникам швидко створювати потужні та масштабовані мобільні додатки з різноманітною функціональністю та зручним управлінням.

# 2 СТВОРЕННЯ ПРОЕКТУ ANDROID STUDIO ТА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ДОДАТКУ

## 2.1 Створення проекту Android Studio

Android Studio — це офіційне інтегроване середовище розробки (IDE) для Android, створене компанією Google. Воно забезпечує розробників усіма необхідними інструментами для створення, тестування та розгортання додатків для Android.

Основні компоненти додатків:

— activity (активність), що представляє один екран користувацького інтерфейсу, але кожен мобільний додаток може містити більше ніж одну активність;

— service (сервіс), який виконує фонові операції, що не потребують взаємодії з користувачем;

— broadcast receiver (широкомовний приймач), що дозволяє додаткам отримувати та обробляти широкомовні повідомлення від системи або інших додатків;

— content provider (постачальник контенту), який дозволяє додаткам зберігати і передавати дані іншим додаткам;

— файл маніфесту, що являє собою файл AndroidManifest.xml, який описує основні характеристики додатку, включаючи його компоненти (активності, сервіси тощо), а також права доступу до ресурсів пристрою.

Структура проекту:

— src/main/java/(пакет) вміщує всі вихідні файли коду (Java або Kotlin), в яких реалізується функціонал додатку та налаштовується розмітка активностей додатку;

— res це папка ресурсів, де зберігаються файли розмітки, зображення, строки та інші ресурси;

— в build.gradle знаходяться скрипти для побудови проекту, які керують налаштуваннями компіляції і залежностями.

Розмітка інтерфейсу користувача:

— Всі макети розміщуються в папці res/layout у вигляді XML файлів розмітки, які використовуються для визначення користувацького інтерфейсу;

— взаємодія з UI відбувається за допомогою методу findViewById, де можна знайти елементи інтерфейсу, визначені у XML файлах, і взаємодіяти з ними у коді Java або Kotlin.

На рисунку 2.1 зображено перші дії для створення нового проекту Android Studio.

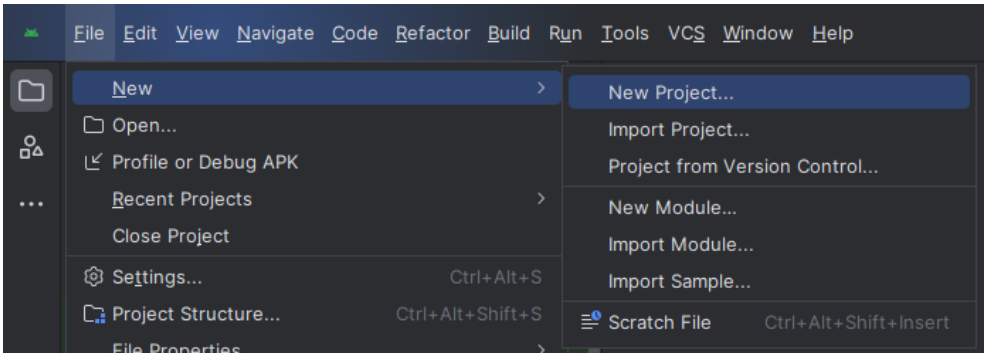


Рисунок 2.1 — Перші дії для створення проекту Android Studio

Далі у відповідному вікні обираємо «Empty Views Activity» щоб проект не мав непотрібних шаблонів, та нажимаємо кнопку «Next» (рис. 2.2).

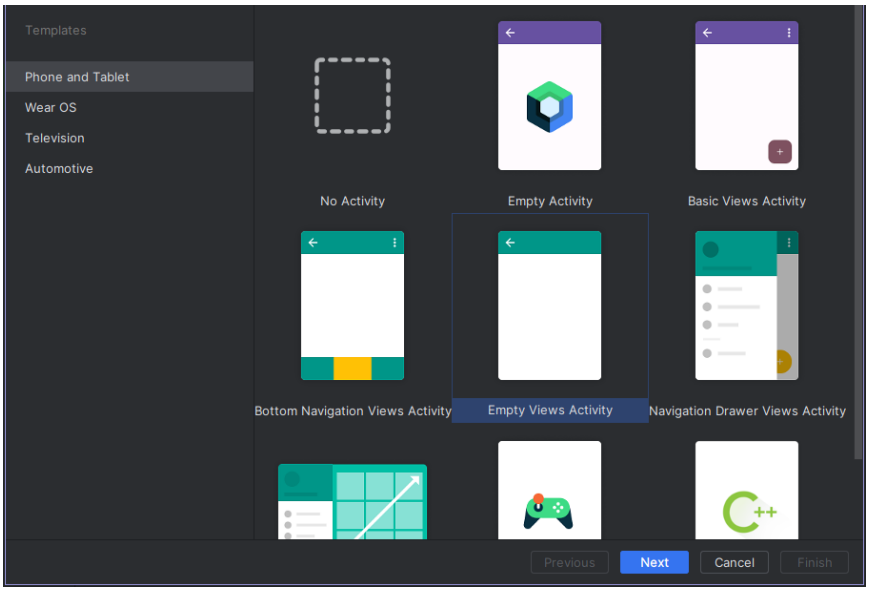


Рисунок 2.2 — Вибір «пустої» активності

Наступним кроком потрібно дати назву проекту та пакету, вибрати мову програмування (Java або Kotlin) та локацію де буде зберігатись проект. Після цього можна завершувати створення натисненням на кнопку «finish» (рис. 2.3).

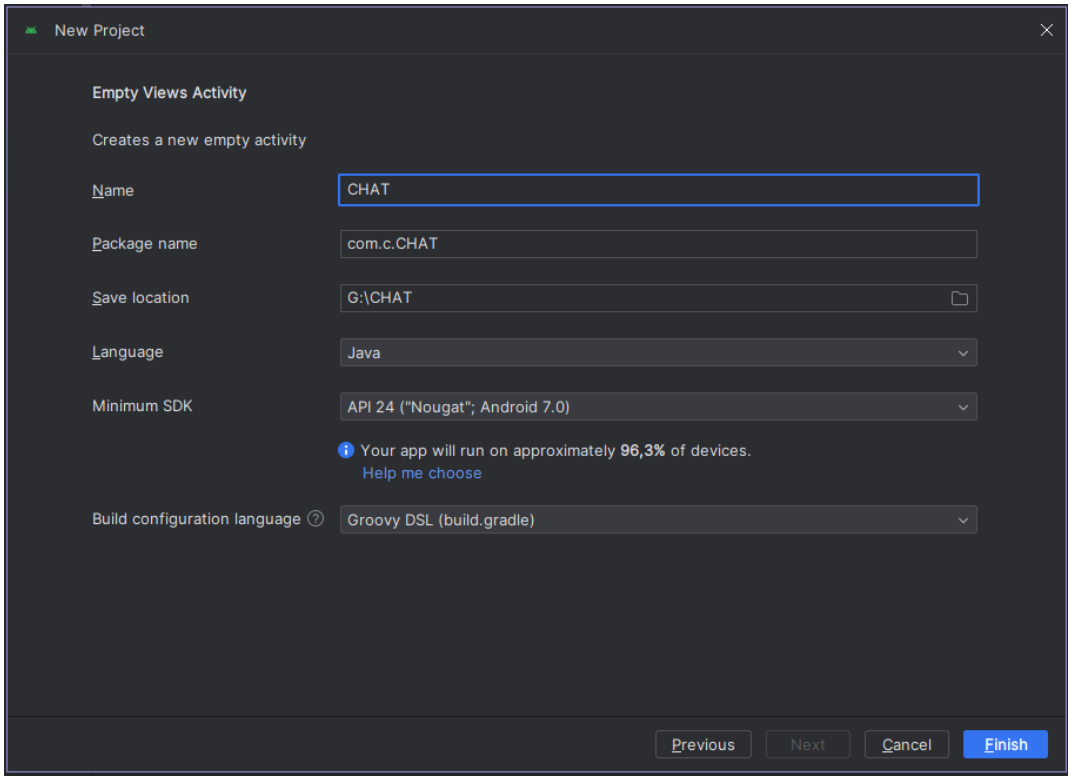


Рисунок 2.3 — Фінальні кроки у створенні проекту

В результаті отримано фундамент майбутнього додатку для оперативної комунікації у вигляді створеного проекту Android Studio. Додаток буде реалізовуватись традиційною мовою для написання мобільних додатків — Java.

## 2.2 Інтеграція Firebase з проектом Android Studio

Firebase — це набір інструментів для розробників мобільних та веб-додатків, який надає різноманітні функції, такі як аналітика, база даних в реальному часі, аутентифікація, хмарне зберігання та багато іншого. Інтеграція Firebase в Android додаток дозволяє легко використовувати ці сервіси для створення потужних і функціональних додатків.

Для початку потрібно вибрати пункт «Firebase» у вкладці меню «Tools» (рис. 2.4)

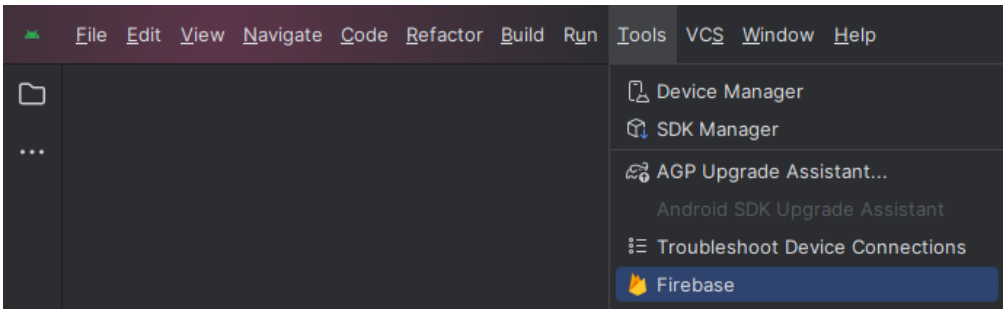


Рисунок 2.4 — Вибір «Firebase» у вкладці меню «Tools»

Наступним кроком потрібно, у відповідній вкладці, розпочати інтегрування Firebase з проектом Android Studio (рис. 2.5).

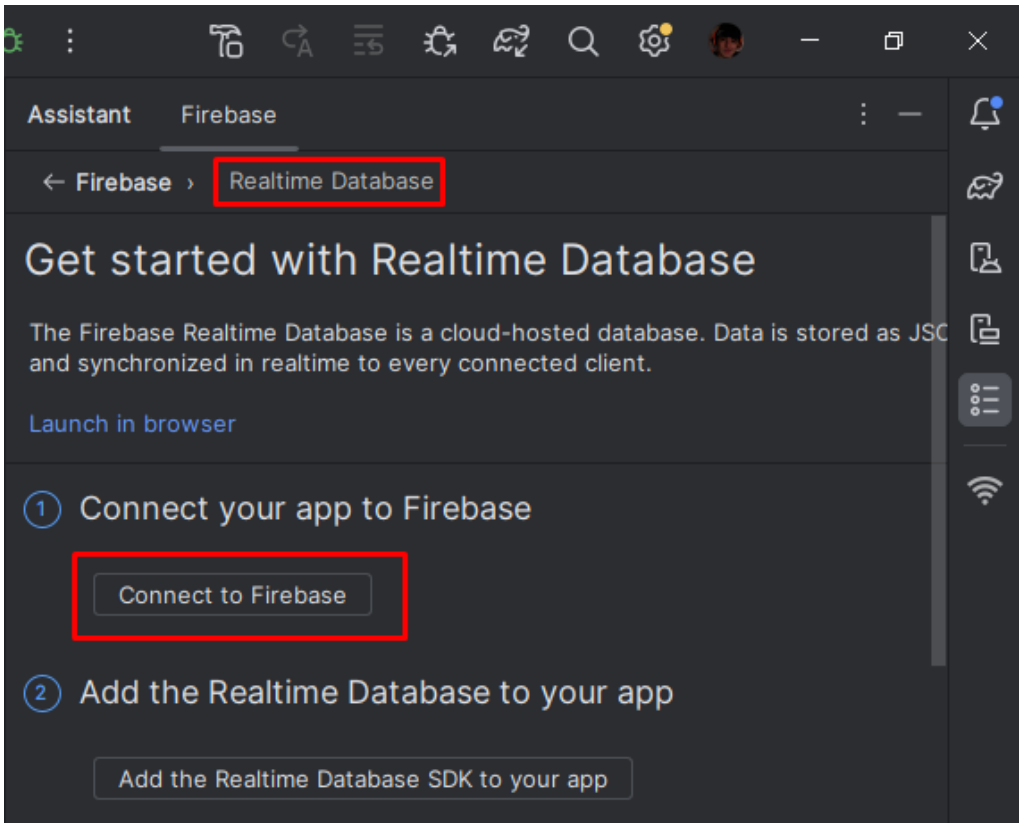


Рисунок 2.5 — Перший етап інтеграції з Firebase

Далі відбувається перенаправлення до «Firebase console» де необхідно створити проект Firebase. На цьому етапі потрібно просто слідувати вказівкам при створенні проекту.

Після створення проекту Firebase на екрані з’являється діалогове вікно в якому тиснемо «Connect» (рис. 2.6).

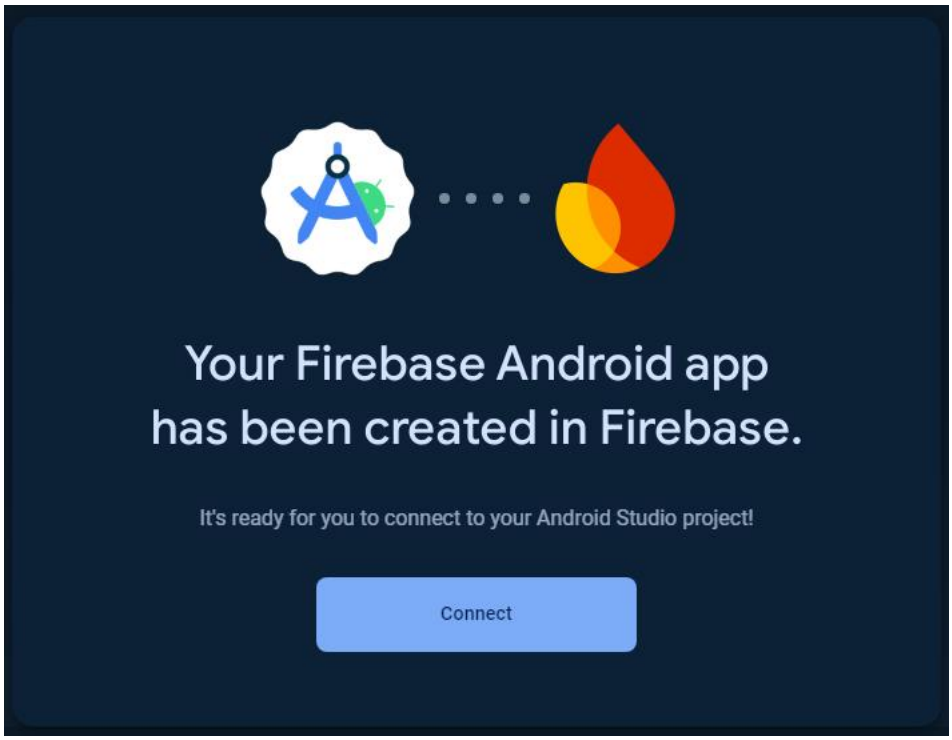


Рисунок 2.6 — Другий етап інтеграції з Firebase

Тепер проект Android Studio успішно інтегровано з проектом Firebase. Про це свідчить вікно яке зображено на рисунку 2.7.

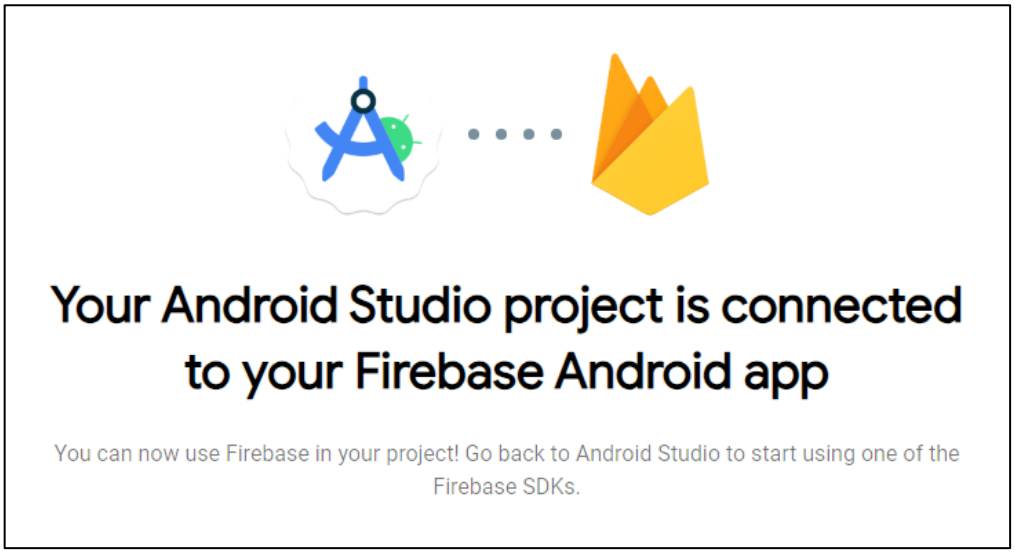


Рисунок 2.7 — Успішне завершення інтеграції проекту Android Studio з Firebase

Далі необхідно інтегрувати Firebase Realtime Database, що дозволяє зберігати та синхронізувати дані в реальному часі (рис 2.8).

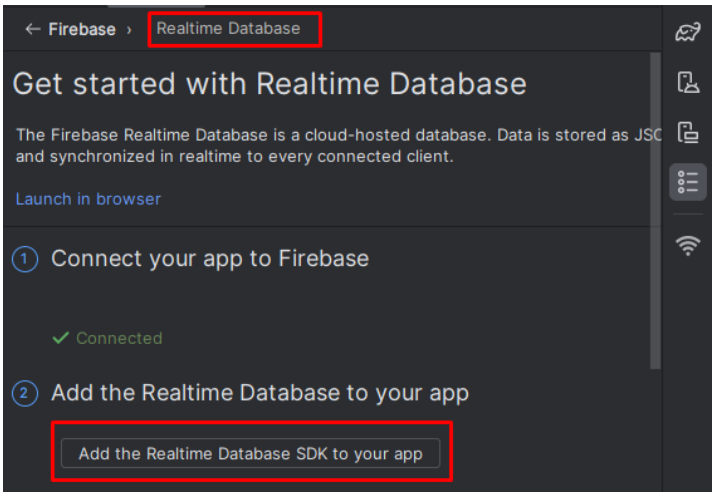


Рисунок 2.8 — Перший етап інтеграції Firebase Realtime Database

Після цього з’являються інструкції, де вказано локації файлів проекту де потрібно вставити відповідний код, який надається в цих ж інструкціях (рис 2.9).

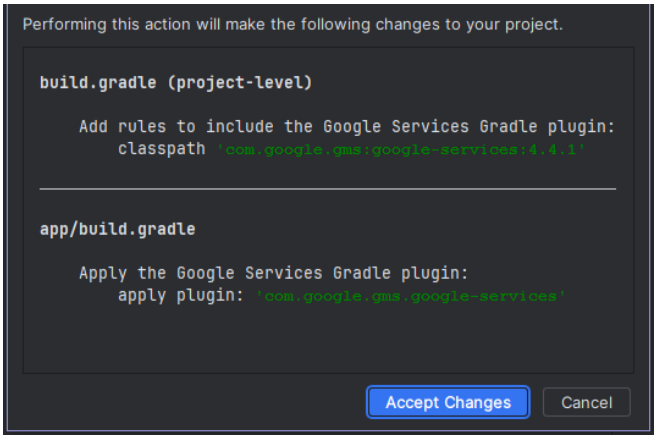


Рисунок 2.9 — Другий етап інтеграції Firebase Realtime Database

Тепер Firebase інтегровано з проектом Android Studio, для необхідних маніпуляцій в майбутній розробці додатку за індивідуальним завданням.

## 2.3 Реалізація екрану запуску (splash screen)

Екран запуску (splash screen) - це екран, який відображається користувачу під час запуску додатку, перед тим, як основний вміст додатку буде повністю завантажено і готовий до використання. Основна його функція — це надати користувачу візуальний сигнал того, що додаток запускається, і показати його

логотип чи інші брендові елементи, що допомагає у зміцненні ідентичності бренду серед користувачів [14]. Іноді екран запуску використовується для виконання підготовчих операцій, таких як завантаження необхідних ресурсів або перевірка підключення до Інтернету, що може збільшити ефективність додатка після його повного завантаження.

Для початку реалізації екрану запуску необхідно створити файл розмітки, назвемо його «activity_splash» в форматі xml. Це потрібно для того, щоб розмістити елементи на екрані, присвоїти їм ідентифікатори та параметри відображення. Процес створення файлу зображений на рисунку 2.10.

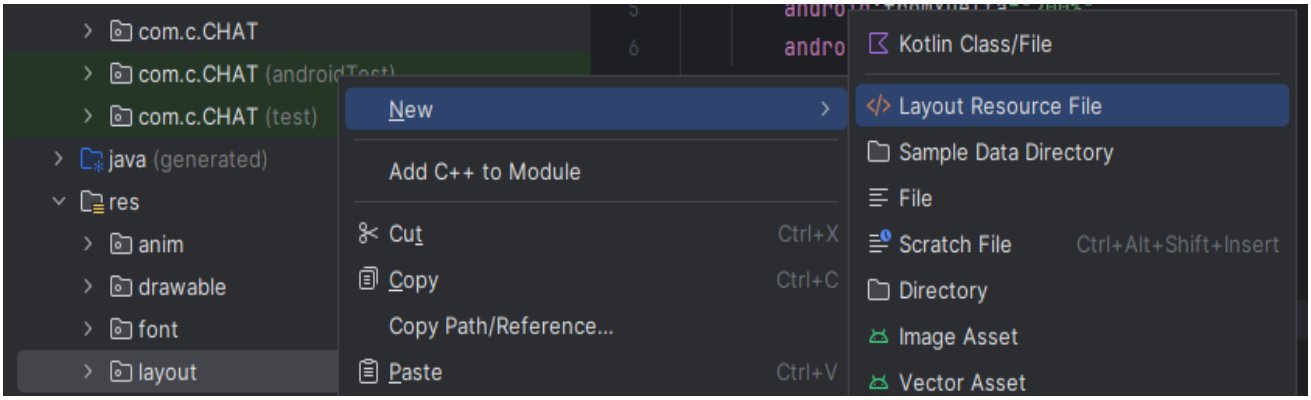


Рисунок 2.10 — Процес створення файлу розмітки

Далі йде присвоєння назви файлу у відповідному полі та закінчення процесу створення файлу розмітки.

Наступним кроком є розміщення елементів, а саме трьох текстових полів (TextView) та однієї картинки (ImageView), яка завчасно була створена за допомогою редактора Adobe Photoshop. Adobe Photoshop — це програмне забезпечення для редагування графіки, що розробляється та випускається компанією Adobe Inc. Вважається одним з найпопулярніших інструментів для обробки зображень у світі. Photoshop використовується для широкого спектру завдань, включаючи ретушування фотографій, створення графічних дизайнів, малювання, створення макетів веб-сторінок та інше.

Після налаштування всіх необхідних параметрів, розмітка activity_splash.xml зображена на рисунку 2.11.


Рисунок 2.11 — Вигляд розмітки activity_splash.xml

Далі створимо клас «splash» в форматі java. Процес створення класу зображений на рисунку 2.12.


Рисунок 2.12 — Процес створення класу

Після цього присвоюється назва класу та закінчується його створення на відповідну кнопку, аналогічно до створення файлу розмітки.

Клас splash унаслідований від AppCompatActivity, що робить його активністю у додатку. В методі onCreate встановлюється вміст екрану за допомогою setContentView з посиланням на activity_splash.xml (ліст. 2.1).

Лістинг 2.1 встановлення вмісту екрану

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_splash);
```

Перед тим як продовжити створимо ще два файли (top_animation.xml, bottom_animation.xml) аналогічно, як було створено activity_splash (рис. 2.10). Вони потрібні для того, щоб додати інтерактивності у вигляді анімацій. Код цих файлів показаний на лістингу 2.2 та лістингу 2.3.

Лістинг 2.2 код файлу top_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<translate
android:fromXDelta="0%"
android:fromYDelta="-200%"
android:duration="2000"/>
<alpha
android:fromAlpha="0.1"
android:toAlpha="1.0"
android:duration="1500"/>
</set>
```

Лістинг 2.3 код файлу bottom_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<translate
android:fromXDelta="0%"
android:fromYDelta="200%"
android:duration="2000"/>
<alpha
android:fromAlpha="0.1"
android:toAlpha="1.0"
android:duration="1500"/>
</set>
```

Кожен файл визначає анімацію, як набір анімаційних дій. У top_animation.xml: translate — елементи зсовуються зверху вниз. Alpha — зміна прозорості елементів з невидимого (0.1) до повністю видимого (1.0). У bottom_animation.xml аналогічно, але елементи зсовуються знизу вгору.

Маючи всі необхідні файли тепер є можливість реалізувати їх використання в класі splash. Розпочнемо з створення змінних та присвоєння їм відповідних значень (ліст. 2.4). Було знайдено всі елементи за їх ідентифікаторами методом findViewById. Далі завантажуються анімації з файлу ресурсів R.anim.top_animation та R.anim.bottom_animation. Анімації встановлюються для кожного відповідного елемента за допомогою setAnimation.

Лістинг 2.4 створення змінних та їх ініціалізація

```
public class splash extends AppCompatActivity {
ImageView logo;
TextView name, own1, own2;
Animation topAnim, bottomAnim;
@SuppressLint("MissingInflatedId")
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_splash);
```

```
getSupportActionBar().hide();
logo = findViewById(R.id.logoimg);
name = findViewById(R.id.logonameimg);
own1 = findViewById(R.id.ownone);
own2 = findViewById(R.id.owntwo);
topAnim = AnimationUtils.loadAnimation(this, R.anim.top_animation);
bottomAnim = AnimationUtils.loadAnimation(this, R.anim.bottom_animation);
```

Далі використовується Handler для затримки на три секунди перед переходом до іншої активності (ліст. 2.5).

Лістинг 2.5 реалізація функції переходу на іншу активність з затримкою

```
new Handler().postDelayed(new Runnable() {
@Override
public void run() {
FirebaseAuth auth = FirebaseAuth.getInstance();
Intent intent;
if (auth.getCurrentUser() != null) {
intent = new Intent(splash.this, MainActivity.class);
} else {
intent = new Intent(splash.this, login.class);
}
startActivity(intent);
finish();
}
}, 3000);
```

Загалом, екран запуску — це важливий елемент дизайну додатків, який не лише відображається при запуску додатку, а й відіграє важливу роль у створенні позитивного враження про нього в очах користувачів.

## 2.4 Реалізація активності входу (login)

Активність входу — це інтерфейс, який використовується для введення облікових даних користувача, таких як ім'я користувача та пароль, з метою отримання доступу до певної системи, додатку або веб-сайту [15].

Активність входу використовується для перевірки ідентичності користувача, запитуючи його облікові дані. Це дозволяє системі або додатку перевірити, чи має користувач доступ до відповідних ресурсів. Введення облікових даних на екрані логіна допомагає забезпечити захист доступу до конфіденційної інформації або функцій, що доступні лише авторизованим користувачам.

Для початку створимо файл розмітки аналогічно тому, як ми це зробили на рисунку 2.10 та назвемо його «activity_login». Далі розмістимо на активності всі необхідні елементи, а саме чотири текстові поля (TextView), одну картинку (ImageView), два поля для введення тексту (EditText) та одну кнопку (Button). Картинка у вигляді назви додатку та логотипу була завчасно підготовлена за допомогою Adobe Photoshop. Після присвоєння ідентифікаторів елементам та налаштування їх параметрів, вигляд активності входу зображена на рисунку 2.13.


Рисунок 2.13 — Вигляд активності входу

Наступним кроком створимо клас з назвою «login» та форматом java, так як зображено на рисунку 2.12.

В класі створюємо змінні та присвоюємо їм значення елементів, які знаходяться на активності за допомогою методу findViewById. Також створюємо progressDialog з текстом «Please wait…», який буде з’являтись при завантаженні інших активностей (ліст. 2.6 та ліст. 2.7).

Лістинг 2.6 створення змінних

```
public class login extends AppCompatActivity {
TextView logsignup;
Button button;
EditText email, password;
FirebaseAuth auth;
String emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+";
android.app.ProgressDialog progressDialog;
```

Лістинг 2.7 ініціалізація змінних

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_login);
getSupportActionBar().hide();
progressDialog = new ProgressDialog(this);
progressDialog.setMessage("Please wait...");
progressDialog.setCancelable(false);
auth = FirebaseAuth.getInstance();
button = findViewById(R.id.logbutton);
email = findViewById(R.id.editTexLogEmail);
password = findViewById(R.id.editTextLogPassword);
logsignup = findViewById(R.id.logsignup);
```

Як показано вище на рисунку, створено ще змінні auth та emailPattern, які в подальшому будуть використовуватись для реалізації функціоналу додатку:

— auth це змінна типу FirebaseAuth, що є класом у Firebase Authentication API, який використовується для керування аутентифікацією користувачів у додатках Android;

— emailPattern це рядок, який містить регулярний вираз для перевірки електронних адрес (email). Регулярні вирази використовуються для пошуку або маніпуляції з текстом, а в даному випадку — для валідації формату email-адреси.

Після цього можна додати обробник подій для кнопки «LOGIN», в якому для початку реалізуємо валідацію даних, за допомогою оператора if (ліст. 2.8).

Лістинг 2.8 реалізація умов виконання авторизації

```
button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
String Email = email.getText().toString();
String pass = password.getText().toString();
if (TextUtils.isEmpty(Email) && TextUtils.isEmpty(pass)){
progressDialog.dismiss();
Toast.makeText(login.this, "Enter the data", Toast.LENGTH_SHORT).show();
}else if (TextUtils.isEmpty(pass)){
progressDialog.dismiss();
Toast.makeText(login.this, "Enter the password",
Toast.LENGTH_SHORT).show();
}else if (TextUtils.isEmpty(Email)) {
progressDialog.dismiss();
Toast.makeText(login.this, "Enter the email", Toast.LENGTH_SHORT).show();
}else if (!Email.matches(emailPattern)){
progressDialog.dismiss();
email.setError("Give correct email");
```

```
}else if (password.length()<6){
progressDialog.dismiss();
password.setError("More then 6 characters");
```

Як показано вище на рисунку, після натиснення на кнопку «LOGIN» створюються дві змінні Email та pass, які ініціалізуються введеними, в поля EditText, електронною поштою та паролем відповідно. Далі уже реалізуються умови виконання авторизації, перша з яких це наявність тексту в полях. Тобто якщо поля для введення електронної пошти та паролю не будуть мати в собі ніякого вмісту, то під час натиснення на кнопку «LOGIN» буде виводитись Toast з текстом «Enter the data». Toast — це клас у Android, який використовується для показу коротких повідомлень користувачеві. Повідомлення з'являється на екрані на кілька секунд і автоматично зникає. Друга та третя умови це уже перевірки на наявність тексту тільки в одному полі. Тобто якщо поле паролю буде пустим а поле електронної пошти наповнене коректним записом то виведеться Toast з текстом «Enter the password». Якщо навпаки, поле електронної пошти буде пустим а поле паролю ні, то виведеться Toast з текстом «Enter the email». В четвертій умові використовується змінна emailPatten для перевірки коректності введеної електронної пошти. Тобто якщо вона введена некоректно то виводиться помилка з текстом «Give correct email». П’ята умова перевіряє довжину введеного паролю. Тобто якщо поле з паролем буде мати менше шести символів то виведеться помилка з текстом «More then 6 characters».

Якщо введенні в поля дані будуть коректними, то виконається код аутентифікації, який прописаний в лістингу 2.9.

Лістинг 2.9 реалізація аутентифікації

```
}else{
auth.signInWithEmailAndPassword(Email,pass).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
@Override
public void onComplete(@NonNull Task<AuthResult> task)
```

```
{
if (task.isSuccessful()){
progressDialog.show();
try
{
Intent intent = new Intent(login.this , MainActivity.class);
startActivity(intent);
finish();
}catch (Exception e){
Toast.makeText(login.this, e.getMessage(), Toast.LENGTH_SHORT).show();}
}else {
Toast.makeText(login.this, task.getException().getMessage(),
Toast.LENGTH_SHORT).show();
}}});}
```

Метод signInWithEmailAndPassword використовується для аутентифікації користувача за допомогою електронної пошти та пароля. Email і pass — це рядки, які містять електронну адресу та пароль відповідно. Метод addOnCompleteListener додає слухача, який буде викликаний після завершення завдання (у даному випадку після спроби входу). Метод onComplete виконується, коли завдання завершено. Аргумент task містить результат завдання, включаючи успішність або невдачу. ProgressDialog.show() показує діалогове вікно з індикатором прогресу, яке інформує користувача про те, що виконується дія. Блок try-catch використовується для обробки можливих виключень при спробі переходу до іншої активності. Далі створюється намір (intent) для запуску MainActivity та за допомогою startActivity(intent) виконується запуск MainActivity.

Загалом, активність входу є важливою складовою для забезпечення безпеки та ідентифікації користувачів у системах, додатках та веб-сайтах. Він допомагає захистити конфіденційні дані та ресурси та забезпечити особисті налаштування і комфорт користувачів.

## 2.5 Реалізація активності реєстрації (registration)

Активність реєстрації — це екран або сторінка в мобільному додатку, де нові користувачі можуть створити обліковий запис, надавши необхідні дані, такі як електронна пошта, пароль та іншу особисту інформацію. Це один з перших і важливих кроків для залучення нових користувачів.

Активність реєстрації необхідна для:

— дозволяє новим користувачам зареєструватися в додатку, що дає їм можливість використовувати функції, доступні лише зареєстрованим користувачам;

— збирає необхідну інформацію користувачів для подальшого використання, наприклад, для персоналізації контенту, відправки сповіщень, тощо;

— забезпечення захищеного доступу до додатку шляхом створення унікальних облікових записів з паролями;

— дозволяє додатку зберігати і використовувати особисті дані користувачів для покращення їхнього досвіду, наприклад, зберігання уподобань, історії використання, тощо.

Для початку реалізації активності реєстрації створимо файл розмітки, з назвою «activity_registration» в форматі xml та java class з назвою «registration», аналогічно тому як це відбувалось з попередніми активностями.

Тепер маючи, хоч і не працюючу але активність реєстрації, потрібно реалізувати перехід від активності входу до активності реєстрації (ліст. 2.10).

Лістинг 2.10 реалізація переходу до активності реєстрації

```
logsignup.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v)
{
Intent intent
```

```
intent = new Intent(login.this,registration.class);
startActivity(intent);
finish();
}});
```

При натисненні на текст «Sign up», який знаходиться в активності входу, тепер буде виконуватись перехід на активність реєстрації.

Наступним кроком потрібно налаштувати розмітку активності реєстрації в якій будуть присутні такі елементи як п’ять текстових полів (TextView), чотири поля для введення тексту (EditText), дві картинки (ImageView) та одна кнопка (Button). Після розміщення всіх елементів та присвоєння їм ідентифікаторів, активність реєстрації має вигляд зображений на рисунку 2.14.

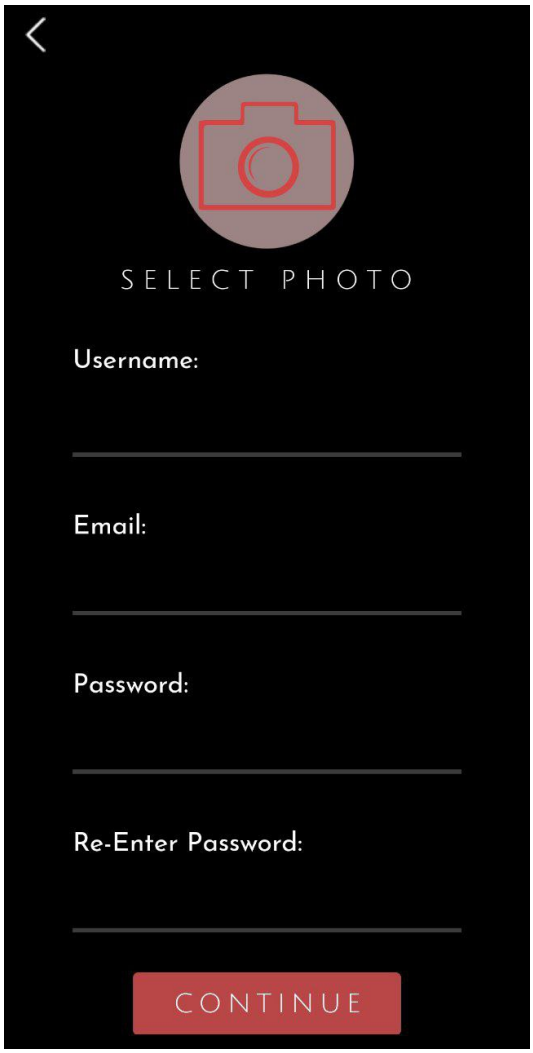


Рисунок 2.14 — Вигляд активності реєстрації

Перейдемо до класу «registration», створимо там змінні та ініціалізуємо їх ідентифікаторами елементів інтерфейсу (ліст 2.11).

Лістинг 2.11 ініціалізація змінних в класі «registration»

```
database = FirebaseDatabase.getInstance();
storage = FirebaseStorage.getInstance();
auth = FirebaseAuth.getInstance();
loginbut = findViewById(R.id.loginbut);
rg_username = findViewById(R.id.rgusername);
rg_email = findViewById(R.id.rgemail);
rg_password = findViewById(R.id.rgpassword);
rg_repassword = findViewById(R.id.rgrepassword);
rg_profileImg = findViewById(R.id.profilerg0);
rg_signup = findViewById(R.id.signupbutton);
```

Як показано вище на рисунку є змінні database та storage:

— FirebaseDatabase.getInstance() це рядок, який отримує екземпляр Firebase Realtime Database, що використовується для збереження та читання даних у реальному часі;

— FirebaseStorage.getInstance() це рядок, який отримує екземпляр Firebase Storage, що використовується для збереження та отримання файлів (наприклад, зображень профілю).

Далі реалізуємо повернення до активності входу, при натисненні на відповідну картинку (ліст. 2.12).

Лістинг 2.12 реалізація повернення до активності входу

```
loginbut.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(registration.this,login.class);
startActivity(intent);
```

```
finish();
}});
```

Наступним кроком реалізуємо обробник подій а саме натискання кнопки реєстрації (ліст. 2.13). Для початку при натисненні на кнопку реєстрації відбуватиметься зчитування введених даних користувачем, а саме ім'я користувача, email, пароль, підтвердження паролю.

Далі відбуватиметься валідація введених даних:

— перевірка, чи всі поля заповнені;

— перевірка коректності email;

— перевірка довжини паролю;

— перевірка збігу паролю та підтвердження паролю.

Лістинг 2.13 реалізація зчитування та валідації даних

```
rg_signup.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
String namee = rg_username.getText().toString();
String emaill = rg_email.getText().toString();
String Password = rg_password.getText().toString();
String cPassword = rg_repassword.getText().toString();
String status = "You can write me";
if (TextUtils.isEmpty(namee) || TextUtils.isEmpty(emaill) ||
TextUtils.isEmpty(Password) || TextUtils.isEmpty(cPassword)){
progressDialog.dismiss();
Toast.makeText(registration.this, "Please enter valid information",
Toast.LENGTH_SHORT).show();
}else  if (!emaill.matches(emailPattern)){
progressDialog.dismiss();
rg_email.setError("Type a valid email");
}else if (Password.length()<6){
```

```
progressDialog.dismiss();
rg_password.setError("Password must be 6 characters or more");
}else if (!Password.equals(cPassword)){
progressDialog.dismiss();
rg_repassword.setError("The password doesn't match");
```

Для продовження роботи необхідно перейти в консоль Firebase, вибрати наш проект, у меню вибрати «Authentication» та натиснути «Get started» та слідувати інструкціям (рис. 2.15).

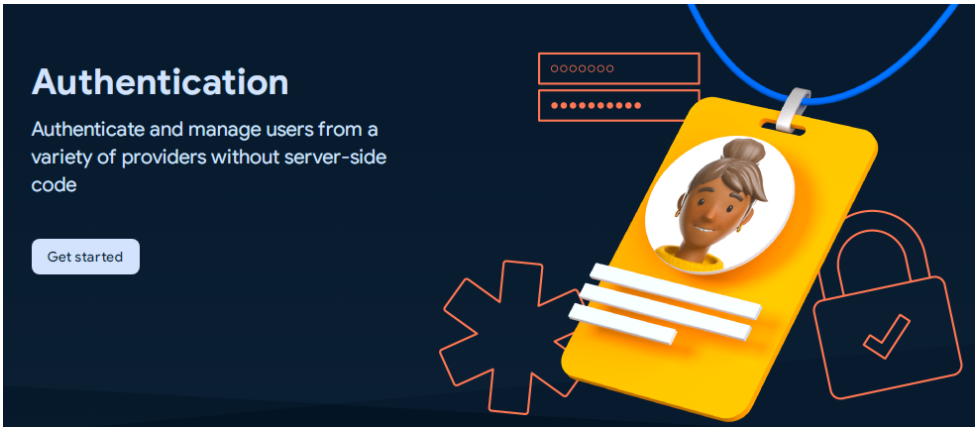


Рисунок 2.15 — Включення аутентифікації

Аналогічні дії виконуються для створення бази даних у вкладці «Realtime Database» та ініціалізації Firebase Storage у вкладці «Storage».

Firebase Storage — це потужне рішення для зберігання файлів, яке забезпечує надійне збереження та швидкий доступ до файлів, а також інтеграцію з іншими сервісами Firebase, такими як Firebase Authentication і Firebase Realtime Database.

Тепер необхідно створити клас «Users», який буде використовуватися для зберігання та обробки інформації про користувачів у додатку (ліст. 2.14).

Лістинг 2.14 створення змінних та конструкторів класу «Users»

```
public class Users {
String profilepic,mail,userName,password,userId,status;
```

```
public  Users(){}
public Users(String userId, String userName, String maill, String password,
String profilepic, String status) {
this.userId = userId;
this.userName = userName;
this.mail = maill;
this.password = password;
this.profilepic = profilepic;
this.status = status; }
```

Ці змінні будуть зберігати такі дані:

— profilepic зберігає URL-адресу профілю зображення користувача;

— mail зберігає електронну адресу користувача;

— userName зберігає ім'я користувача;

— password зберігає пароль користувача;

— userId зберігає унікальний ідентифікатор користувача;

— status зберігає статус користувача (наприклад, "You can write me").

Порожній конструктор потрібен для деяких фреймворків (наприклад, Firebase), щоб мати можливість створювати екземпляри класу без параметрів. Після нього йде конструктор, який ініціалізує всі поля класу значеннями, переданими як параметри. Далі прописуємо методи доступу (гетери та сетери), після чого можна вважати, що клас «Users» повністю реалізовано.

Тепер реалізуємо вибір зображення, яке буде являти собою фото профілю користувача (ліст. 2.15).

Лістинг 2.15 реалізація вибору зображення профілю користувача

```
rg_profileImg.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent();
intent.setType("image/*");
```

```
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent,"Select Picture"),10);
}});}
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable
Intent data) {
super.onActivityResult(requestCode, resultCode, data);
if (requestCode==10){
if (data!=null){
imageURI = data.getData();
rg_profileImg.setImageURI(imageURI);}}}
```

При кліку на елемент rg_profileImg, виконується код, що знаходиться всередині методу onClick. У цьому коді створюється intent для вибору зображення (Intent.ACTION_GET_CONTENT), із заданим типом даних "image/*". Після цього викликається метод startActivityForResult, який відкриває діалогове вікно вибору зображення для користувача. Параметр «10» передається як requestCode, щоб пізніше в іншому методі можна було ідентифікувати результат, повернений цим запитом. Коли користувач обирає зображення у діалоговому вікні та завершує вибір, метод onActivityResult викликається автоматично. В цьому методі перевіряється, чи requestCode дорівнює 10, що підтверджує, що це результат вибору зображення. Якщо data (дані, що повертаються) не є нульовими, отримується URI обраного зображення і встановлюється для елемента rg_profileImg, щоб відобразити обране зображення у профільному зображенні користувача.

Виконавши попередні кроки тепер є можливість реалізувати створення нового облікового запису користувача в системі аутентифікації Firebase та збереження інформації про користувача в базі даних Firebase Realtime Database. Після перевірки валідності даних додаємо оператор else, щоб код виконався якщо всі дані є валідними, та викликаємо метод createUserWithEmailAndPassword

об'єкта auth, який створює нового користувача за допомогою переданих електронної пошти (emaill) та пароля (Password). До цього методу приєднано слухача подій OnCompleteListener, який викликається, коли операція створення облікового запису завершується. Далі перевіряється, чи було успішно створено обліковий запис користувача. Якщо так, то отримується унікальний ідентифікатор (uid) для нового користувача, який використовується для створення посилань на його дані у базі даних та сховищі Firebase. Створюється посилання на розділ бази даних «user», де будуть зберігатися дані про користувачів, та на розділ сховища «Upload», де будуть зберігатися зображення користувачів (ліст. 2.16).

Лістинг 2.16 створення посилань на розділи «user» та «Upload»

```
auth.createUserWithEmailAndPassword(emaill,Password)
.addOnCompleteListener(new OnCompleteListener<AuthResult>() {
@Override
public void onComplete(@NonNull Task<AuthResult> task) {
if (task.isSuccessful()){
String id = task.getResult().getUser().getUid();
DatabaseReference reference = database.getReference()
.child("user").child(id);
StorageReference storageReference =
storage.getReference().child("Upload").child(id);
```

Далі перевірятимемо, чи користувач вибрав зображення для свого профілю. Якщо так, виконується код для завантаження зображення до сховища Firebase (ліст. 2.17).

Лістинг 2.17 створення облікового запису з вибраним зображенням профілю

```
if (imageURI!=null){
storageReference.putFile(imageURI).addOnCompleteListener(new
OnCompleteListener<UploadTask.TaskSnapshot>() {
@Override
```

```
public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {
if (task.isSuccessful()){
storageReference.getDownloadUrl().addOnSuccessListener(new
OnSuccessListener<Uri>() {
@Override
public void onSuccess(Uri uri) {
imageuri = uri.toString();
Users users = new Users(id,namee,emaill,Password,imageuri,status);
reference.setValue(users)
.addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()){
progressDialog.show();
Intent intent
intent = new Intent(registration.this,MainActivity.class);
startActivity(intent);
finish();
}else {
Toast.makeText(registration.this, "Error in creating the user",
Toast.LENGTH_SHORT).show();
}}});}});}});
```

Створюється об'єкт Users з даними нового користувача, а потім цей об'єкт зберігається у базі даних Firebase. Якщо операція збереження успішна, користувач перенаправляється на головний екран додатку. Якщо створення облікового запису не вдалося, виводиться сповіщення про помилку з повідомленням про причину невдачі.

Якщо ж користувач не вибрав зображення, то необхідно в консолі Firebase у вкладці «Storage» натиснути на «Upload file» та завантажити зображення, яке буде

використовуватись як зображення за замовчуванням (рис. 2.17). Зображення було завчасно створене за допомогою редактора Adobe Photoshop.

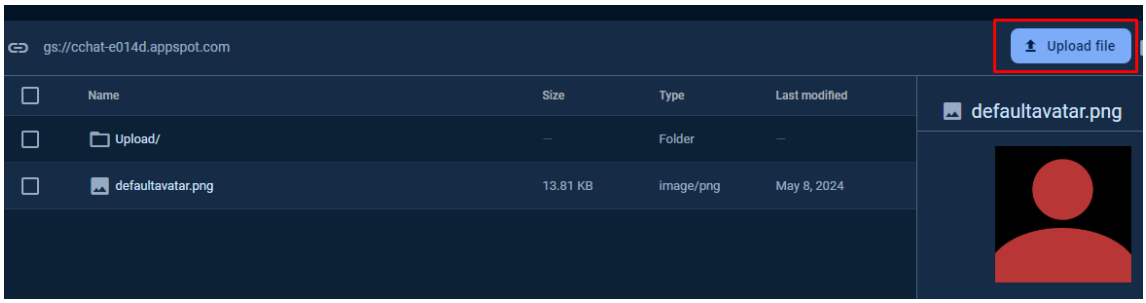


Рисунок 2.17 — Завантаження зображення за замовчуванням

Далі з використанням оператора else реалізуємо створення нового облікового запису користувача який не вибрав зображення профілю, для цього використовується URL зображення за замовчуванням. Інший код залишається незмінним (ліст. 2.18). При спробі зареєструвати нових користувачів, в базі даних Firebase, з’являються записи, які зображенні на рисунку 2.18.

Лістинг 2.18 створення облікового запису без вибраного фото профілю

```
}else {
imageuri = "https://firebasestorage.googleapis.com/v0/b/cchat-e014d.appspot.com/o/defaultavatar.png?alt=media&token=37818586-2538-4597-926c-008a915562f8";
Users users = new Users(id,namee,emaill,Password,imageuri,status);
reference.setValue(users).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()){
progressDialog.show();
Intent intent = new Intent(registration.this,MainActivity.class);
startActivity(intent);
finish();
```

```
}else {
Toast.makeText(registration.this, "Error in creating the user",
Toast.LENGTH_SHORT).show();
}}});}
```

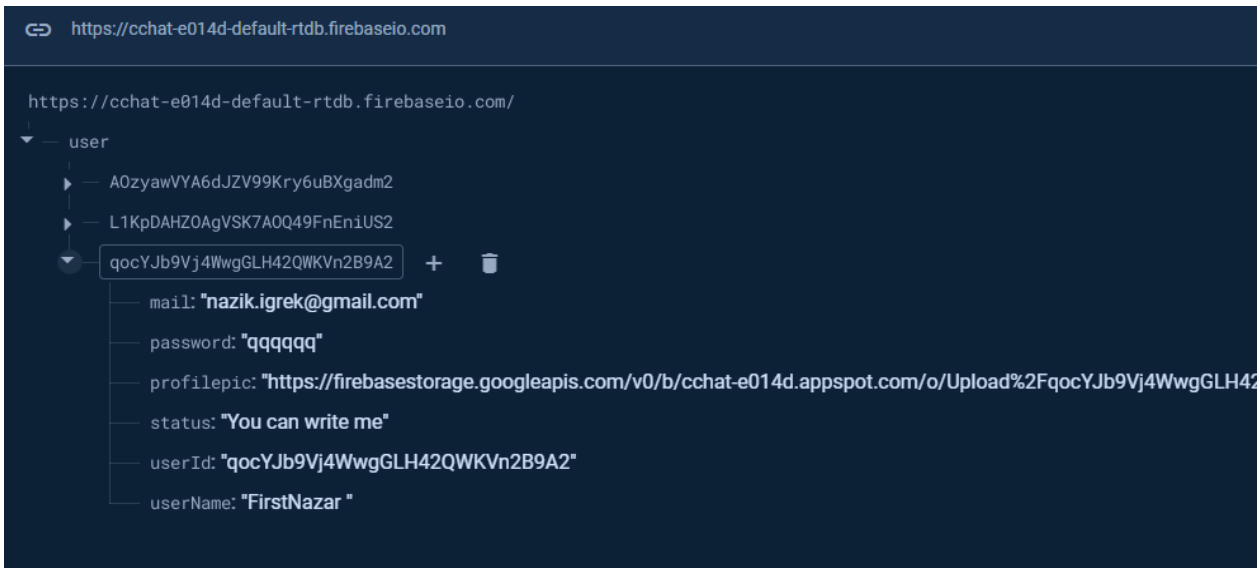


Рисунок 2.18 — Вигляд інформації про користувача в базі даних Firebase

Це говорить про те що активність реєстрації працює правильно та готова до використання. Вона забезпечує належну обробку введених користувачами даних, валідацію інформації та інтеграцію з бекенд-системою для зберігання облікових записів. Крім того, активність успішно управляє помилками та забезпечує користувачам зворотний зв'язок у випадку некоректних даних, що підвищує загальний досвід використання додатка.

## 2.6 Реалізація активності відображення користувачів (MainActivity)

В Android додатках відображення списку користувачів часто використовується у соціальних мережах, чатах, додатках для спільної роботи тощо. Основна мета такої активності — забезпечити зручний та інтуїтивний інтерфейс для перегляду та взаємодії зі списком користувачів.

Для початку реалізації активності відображення користувачів тепер не потрібно створювати файл розмітки та java клас, так як MainActivity.java

створюється автоматично з проектом, що означає наявність і файлу розмітки з назвою «activity_main».

Після розміщення всіх елементів та присвоєння їм ідентифікаторів, зображення головної активності показано на рисунку 2.19.

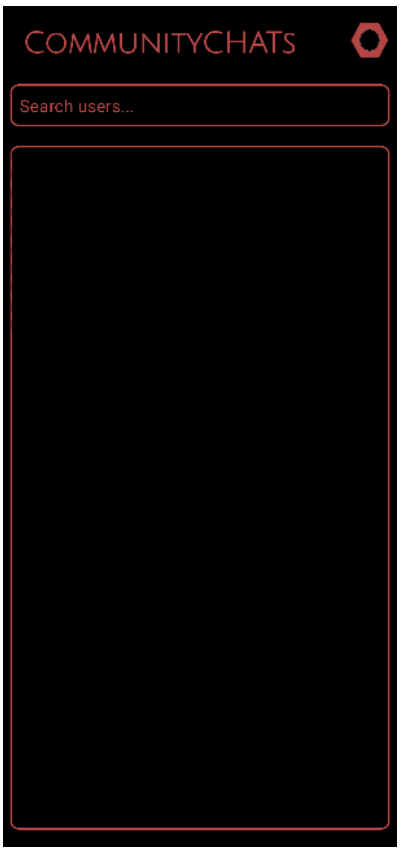


Рисунок 2.19 — Вигляд головної активності

Головним елементом на цій активності є RecyclerView. За допомогою нього буде реалізовано відображення користувачів додатку. Для цього необхідно реалізувати адаптер для користувачів, тому створюємо клас та даємо йому назву «UserAdpter».

В реалізації цього класу створюємо конструктор, який створює копії списків користувачів та буде ініціалізувати адаптер з початковим списком користувачів. Також створюємо метод onCreateViewHolder, який буде викликатись для створення нових елементів RecyclerView та інфлювання нового вигляду макету user_item (ліст. 2.19). Щоб реалізувати user_item створимо з відповідною назвою XML файл та налаштуємо дизайн цього елементу (рис. 2.20).

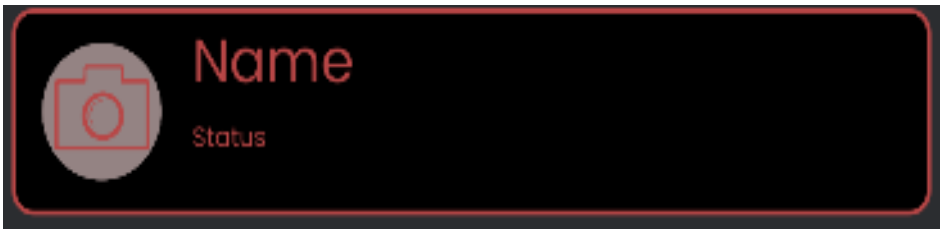


Рисунок 2.20 — Дизайн елемента user_item

Лістинг 2.19 реалізація конструктора UserAdpter та onCreateViewHolder

```
public class UserAdpter extends
RecyclerView.Adapter<UserAdpter.viewholder> {Context mainActivity;
ArrayList<Users> usersArrayList;
ArrayList<Users> filteredList;
public UserAdpter(Context mainActivity, ArrayList<Users>
usersArrayList) {
this.mainActivity = mainActivity;
this.usersArrayList = new ArrayList<>(usersArrayList);
this.filteredList = new ArrayList<>(usersArrayList); }
@NonNull
@Override
public UserAdpter.viewholder onCreateViewHolder(@NonNull
ViewGroup parent, int viewType) {
View view = LayoutInflater.from(mainActivity).inflate(R.layout.user_item,
parent, false);
return new viewholder(view); }
```

Далі потрібно прив'язати дані, тобто встановити текст для імені, статусу користувача та завантажити зображення профілю. Також він буде додавати обробник натискань для переходу до вікна чату. Тому для початку створимо файл розмітки з назвою «activity_chatwindo» та java клас з назвою «chatwindo», реалізація яких буде виконана пізніше. Для прив’язки даних створюємо метод onBindViewHolder, код якого написаний в лістингу 2.20.

Лістинг 2.20 реалізація прив’язки даних та переходу до активності чату

```
@Override
public void onBindViewHolder(@NonNull UserAdpter.viewholder holder, int position) {
Users users = filteredList.get(position);
holder.username.setText(users.getUserName());
holder.userstatus.setText(users.getStatus());
Picasso.get().load(users.getProfilepic()).into(holder.userimg);
holder.itemView.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
Intent intent = new Intent(mainActivity, chatwindo.class);
intent.putExtra("nameeee", users.getUserName());
intent.putExtra("reciverImg", users.getProfilepic());
intent.putExtra("uid", users.getUserId());
mainActivity.startActivity(intent);
}});}
```

Тепер додаємо ще два методи setUsersList та filter (ліст. 2.21).

Лістинг 2.21 код методів setUsersList та filter

```
public void setUsersList(ArrayList<Users> usersArrayList) {
this.usersArrayList.clear();
this.usersArrayList.addAll(usersArrayList);
this.filteredList.clear();
this.filteredList.addAll(usersArrayList);
notifyDataSetChanged();}
public void filter(String text) {
filteredList.clear();
if (text.isEmpty()) {
filteredList.addAll(usersArrayList);
```

```
} else {
for (Users user : usersArrayList) {
if (user.getUserName().toLowerCase().contains(text.toLowerCase())) {
filteredList.add(user);}}}
notifyDataSetChanged();}
```

Метод setUsersList оновлює списки usersArrayList та filteredList новими даними та викликає notifyDataSetChanged для оновлення інтерфейсу.

Метод filter фільтрує користувачів за введеним текстом, очищає filteredList і додає відповідні елементи та викликає notifyDataSetChanged для оновлення інтерфейсу. Ініціалізація змінних та реалізація ValueEventListener показана на лістингу 2.22.

Далі в класі MainActivity створюємо метод onCreate, який:

— викликається при створенні активності;

— ініціалізує Firebase, компоненти інтерфейсу та RecyclerView;

— створюється адаптер для RecyclerView;

— додається ValueEventListener, який викликається коли дані з Firebase змінюються, очищає список користувачів і додає нові дані, окрім поточного користувача та оновлює адаптер;

— додається TextWatcher до searchBar для фільтрації користувачів під час введення тексту (ліст. 2.23).

Лістинг 2.22 ініціалізація змінних та реалізація ValueEventListener

```
database = FirebaseDatabase.getInstance();
auth = FirebaseAuth.getInstance();
setbut = findViewById(R.id.settingBut);
searchBar = findViewById(R.id.searchBar);
DatabaseReference reference = database.getReference().child("user");
usersArrayList = new ArrayList<>();
mainUserRecyclerView = findViewById(R.id.mainUserRecyclerView);
mainUserRecyclerView.setLayoutManager(new LinearLayoutManager(this));
```

```
adapter = new UserAdpter(MainActivity.this, usersArrayList);
mainUserRecyclerView.setAdapter(adapter);
reference.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
usersArrayList.clear();
String currentUserId = auth.getCurrentUser().getUid();
for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
Users users = dataSnapshot.getValue(Users.class);
if (!users.getUserId().equals(currentUserId)) {
usersArrayList.add(users);
}}
adapter.setUsersList(usersArrayList);
adapter.notifyDataSetChanged();}
```

Лістинг 2.23 реалізація методу searchBar

```
searchBar.addTextChangedListener(new TextWatcher() {
@Override
public void beforeTextChanged(CharSequence s, int start, int count, int after) {
}
@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
adapter.filter(s.toString());}
@Override
public void afterTextChanged(Editable s) {
}});
```

Ця активність — головна сторінка додатка. При старті вона завантажує список користувачів, які відображаються у RecyclerView, де кожен елемент містить ім'я, статус та зображення профілю.

## 2.7 Реалізація активності налаштувань (setting)

Для початку, аналогічно тому як це виконувалось з попередніми активностями, створимо файл розмітки з назвою «activity_setting» та java клас з назвою «setting». Тепер реалізуємо перехід з головної активності до активності налаштувань, при натисненні на відповідну кнопку (ліст. 2.24).

Лістинг 2.24 реалізація переходу до активності налаштувань

```
setbut.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(MainActivity.this, setting.class);
startActivity(intent); }});
```

Після налаштування розмітки активності налаштувань, вона має вигляд зображений на рисунку 2.21.

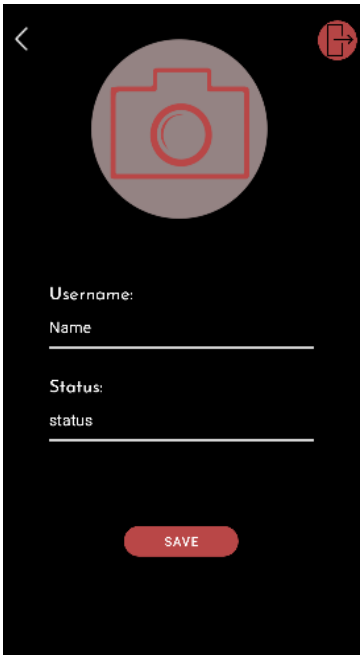


Рисунок 2.21 — Вигляд активності налаштувань

В першу чергу реалізуємо повернення до головної активності, при натисненні на відповідну кнопку (ліст. 2.25).

Лістинг 2.25 реалізація повернення до головної активності

```
mainbutback = findViewById(R.id.mainbutback);
mainbutback.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(setting.this,MainActivity.class);
startActivity(intent);
finish();}});
```

Далі створимо обробник, який відповідає за вихід користувача з системи. При натисканні кнопки виходу, з'являється діалогове вікно, де користувач може підтвердити або скасувати вихід. Якщо вихід підтверджується, виконується вихід з Firebase і перехід до активності входу в систему (ліст. 2.26).

Лістинг 2.26 обробник для виходу користувача з системи

```
imglogout = findViewById(R.id.logoutimg);
imglogout.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Dialog dialog = new Dialog(setting.this,R.style.dialoge);
dialog.setContentView(R.layout.dialog_layout);
Button no,yes;
yes = dialog.findViewById(R.id.yesbnt);
no = dialog.findViewById(R.id.nobnt);
yes.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
FirebaseAuth.getInstance().signOut();
Intent intent = new Intent(setting.this,login.class);
startActivity(intent);
finish();}});
```

```
no.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
dialog.dismiss();}});
dialog.show();}});
```

Наступним кроком створимо метод відслідковує зміни в базі даних Firebase і завантажує інформацію користувача (ім'я, статус та зображення профілю). Інформація відображається у відповідних полях (ліст. 2.27).

Лістинг 2.27 реалізація методу для завантаження інформації

```
DatabaseReference reference =
database.getReference().child("user").child(auth.getUid());
StorageReference storageReference =
storage.getReference().child("Upload").child(auth.getUid());
reference.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
email = snapshot.child("mail").getValue().toString();
password = snapshot.child("password").getValue().toString();
String name = snapshot.child("userName").getValue().toString();
String profile = snapshot.child("profilepic").getValue().toString();
String status = snapshot.child("status").getValue().toString();
setname.setText(name);
setstatus.setText(status);
Picasso.get().load(profile).into(setprofile);
}
@Override
public void onCancelled(@NonNull DatabaseError error) {
}});
```

Тепер створимо обробник, який дозволяє користувачу вибрати нове зображення профілю зі свого пристрою. Відкривається вибір зображення з галереї (ліст. 2.28).

Лістинг 2.28 обробник для вибору нового зображення профілю

```
setprofile.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
Intent intent = new Intent();
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent, "Select Picture"), 10);
}});
```

Останнім кроком буде реалізація обробника, який відповідає за збереження змін, зроблених користувачем. Він показує діалогове вікно прогресу і перевіряє, чи було вибране нове зображення профілю. Якщо так, то нове зображення завантажується в Firebase Storage і зберігається його URL. Далі всі зміни (ім'я, статус, зображення профілю) зберігаються в Firebase Realtime Database. Після успішного збереження, користувач повертається на головний екран (ліст. 2.29).

Лістинг 2.29 обробник, що відповідає за збереження змін

```
donebut.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
progressDialog.show();
String name = setname.getText().toString();
String Status = setstatus.getText().toString();
if (setImageUri!=null){
storageReference.putFile(setImageUri).addOnCompleteListener(new
OnCompleteListener<UploadTask.TaskSnapshot>() {
```

```
@Override
public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {
storageReference.getDownloadUrl().addOnSuccessListener(new
OnSuccessListener<Uri>() {
@Override
public void onSuccess(Uri uri) {
String finalImageUri = uri.toString();
Users users = new Users(auth.getUid(),
name,email,password,finalImageUri,Status);
reference.setValue(users).addOnCompleteListener(new
OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()){
progressDialog.dismiss();
Toast.makeText(setting.this, "Data Is save ", Toast.LENGTH_SHORT).show();
Intent intent = new Intent(setting.this,MainActivity.class);
startActivity(intent);
finish();
}else {
progressDialog.dismiss();
Toast.makeText(setting.this, "error", Toast.LENGTH_SHORT).show();
}}});}});}});
```

Активність налаштувань у додатку призначена для того, щоб користувачі могли редагувати свою особисту інформацію, таку як ім'я, статус та зображення профілю. Це місце, де користувачі можуть налаштувати свій профіль відповідно до власних уподобань та зберегти ці зміни в базі даних Firebase.

## 2.8 Реалізація активності чату (chat window)

Так як файл розмітки та java клас уже створені, необхідно розмістити елементи на цій активності (рис. 2.22). Після цього створимо ще два java класи messagesAdpter та msgModelclass. Адаптер messagesAdpter відповідатиме за відображення повідомлень в RecyclerView. Клас msgModelclass буде моделлю даних для повідомлень.

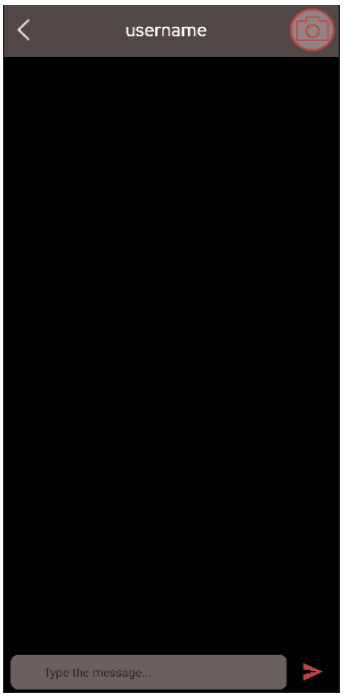


Рисунок 2.22 — Вигляд активності чату

В класі msgModelclass створюємо два конструктори (ліст. 2.30):

— без аргументів для використання Firebase;

— з параметрами для створення нових об'єктів повідомлень.

Лістинг 2.30 конструктори класу msgModelclass

```
public class msgModelclass {
String message;
String senderid;
long timeStamp;
String messageKey;
```

```
public msgModelclass() {}
public msgModelclass(String message, String senderid, long timeStamp) {
this.message = message;
this.senderid = senderid;
this.timeStamp = timeStamp; }
```

Конструктор з параметрами використовується для створення нових об'єктів повідомлень з заданими значеннями тексту повідомлення, UID відправника та мітки часу.

Далі створюємо геттери та сеттери (ліст. 2.31). Це методи для отримання та встановлення значень властивостей повідомлень (текст повідомлення, UID відправника, мітка часу, ключ повідомлення).

Лістинг 2.31 встановлення значень властивостей повідомлень

```
public String getMessage() {
    return message; }
  public void setMessage(String message) {
    this.message = message; }
  public String getSenderid() {
    return senderid;}
  public void setSenderid(String senderid) {
    this.senderid = senderid;}
  public long getTimeStamp() {
    return timeStamp;}
  public void setTimeStamp(long timeStamp) {
    this.timeStamp = timeStamp;}
  public String getMessageKey() {
    return messageKey; }
  public void setMessageKey(String messageKey) {
    this.messageKey = messageKey; }
```

Тепер в класі messagesAdpter реалізуємо метод onCreateViewHolder (ліст 2.32). Він створює та повертає ViewHolder на основі типу повідомлення (відправлене або отримане).

Лістинг 2.32 реалізація методу onCreateViewHolder

```
@NonNull
@Override
public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
if (viewType == ITEM_SEND) {
View view = LayoutInflater.from(context).inflate(R.layout.sender_layout, parent,
false);
return new senderViewHolder(view);
} else {
View view = LayoutInflater.from(context).inflate(R.layout.reciver_layout, parent,
false);
return new reciverViewHolder(view);}}
```

Далі в методі onBindViewHolder прив'язуємо дані до ViewHolder, налаштовуючи текст та зображення профілю відправника/одержувача (ліст. 2.33). Встановлює слухач довгого натискання для видалення повідомлень з бази даних Firebase (ліст. 2.34).

Лістинг 2.33 прив’язка даних до відправника/одержувача

```
if (holder instanceof senderViewHolder) {
senderViewHolder viewHolder = (senderViewHolder) holder;
viewHolder.msgtxt.setText(messages.getMessage());
Picasso.get().load(chatwindo.senderImg).into(viewHolder.circleImageView);
} else {
reciverViewHolder viewHolder = (reciverViewHolder) holder;
viewHolder.msgtxt.setText(messages.getMessage());
```

```
Picasso.get().load(chatwindo.reciverIImg).into(viewHolder.circleImageView); }
```

Лістинг 2.34 реалізація можливості видалення повідомлень

```
holder.itemView.setOnLongClickListener(new View.OnLongClickListener() {
@Override
public boolean onLongClick(View view) {
new AlertDialog.Builder(context)
.setTitle("Delete")
.setMessage("Do you want to delete this message?")
.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialogInterface, int i) {
int adapterPosition = holder.getAdapterPosition();
if (adapterPosition == RecyclerView.NO_POSITION) {
return; }
String messageKey =
messagesAdpterArrayList.get(adapterPosition).getMessageKey();
FirebaseDatabase.getInstance().getReference().child("chats").child(chatwindo.sen
derRoom).child("messages").child(messageKey).removeValue();
FirebaseDatabase.getInstance().getReference().child("chats").child(chatwindo.rec
iverRoom).child("messages").child(messageKey).removeValue();
messagesAdpterArrayList.remove(adapterPosition);
notifyItemRemoved(adapterPosition);
}})
.setNegativeButton("No", (dialogInterface, i) -> {
dialogInterface.dismiss();})
.show();
return true; }});
```

Тепер реалізуємо визначення типу повідомлення (відправлене чи отримане) та внутрішні класи ViewHolder для відправлених та отриманих повідомлень (ліст. 2.35 та ліст. 2.36).

Лістинг 2.35 визначення типу повідомлення

```
@Override
public int getItemViewType(int position) {
msgModelclass messages = messagesAdpterArrayList.get(position);
if
(FirebaseAuth.getInstance().getCurrentUser().getUid().equals(messages.getSende
rid())) {
return ITEM_SEND;
} else {
return ITEM_RECIVE; }}
```

Лістинг 2.36 класи ViewHolder для відправлених та отриманих повідомлень

```
class senderViewHolder extends RecyclerView.ViewHolder {
CircleImageView circleImageView;
TextView msgtxt;
public senderViewHolder(@NonNull View itemView) {
super(itemView);
circleImageView = itemView.findViewById(R.id.profilerggg);
msgtxt = itemView.findViewById(R.id.msgsendertyp);}}
class reciverViewHolder extends RecyclerView.ViewHolder {
CircleImageView circleImageView;
TextView msgtxt;
public reciverViewHolder(@NonNull View itemView) {
super(itemView);
circleImageView = itemView.findViewById(R.id.pro);
msgtxt = itemView.findViewById(R.id.recivertextset);}}
```

Ці класи є ViewHolder-ами для різних типів повідомлень, що визначають, як виглядатимуть і будуть оброблятися окремі елементи в RecyclerView.

Загалом, код забезпечує різне відображення повідомлень у чаті залежно від того, чи було повідомлення відправлене поточним користувачем, чи отримане від іншого користувача. Це дозволяє створити зручний інтерфейс для відображення

чатів, де користувач може легко розрізняти свої повідомлення та повідомлення інших користувачів.

Наступним кроком в класі chatwindo, після створення змінних, в методі onCreate напишемо код, який прописаний в лістингу 2.37. Також реалізуємо повернення до головної активності при натисненні на відповідну кнопку, аналогічним методом, як це відбувалось в попередніх активностях.

Лістинг 2.37 ініціалізація змінних та початок реалізація активності чату

```
database = FirebaseDatabase.getInstance();
firebaseAuth = FirebaseAuth.getInstance();
reciverName = getIntent().getStringExtra("nameeee");
reciverimg = getIntent().getStringExtra("reciverImg");
reciverUid = getIntent().getStringExtra("uid");
messagesArrayList = new ArrayList<>();
sendbtn = findViewById(R.id.sendbtnn);
textmsg = findViewById(R.id.textmsg);
reciverNName = findViewById(R.id.recivername);
profile = findViewById(R.id.profileimgg);
messageAdpter = findViewById(R.id.msgadpter);
LinearLayoutManager linearLayoutManager = new LinearLayoutManager(this);
linearLayoutManager.setStackFromEnd(true);
messageAdpter.setLayoutManager(linearLayoutManager);
mmessagesAdpter = new messagesAdpter(chatwindo.this, messagesArrayList);
messageAdpter.setAdapter(mmessagesAdpter);
Picasso.get().load(reciverimg).into(profile);
reciverNName.setText(reciverName);
SenderUID = firebaseAuth.getUid();
senderRoom = SenderUID + reciverUid;
reciverRoom = reciverUid + SenderUID;
```

В коді відбувається ініціалізація інтерфейсу користувача для вікна чату. Далі отримуємо інстанції Firebase Database, Firebase Auth та дані про одержувача (ім'я, фото, UID). Також відбувається ініціалізація списку повідомлень, адаптера для них та встановлюються зображення профілю з іменем одержувача у відповідні елементи інтерфейсу та унікальні ідентифікатори кімнат для відправника та одержувача.

Після створення посилань на базу даних для користувача та чату, необхідно реалізувати метод onDataChange (ліст. 2.38). Функціонал методу:

— викликається при зміні даних у Firebase Database;
— очищає поточний список повідомлень;
— додає нові повідомлення до списку з отриманих даних;
— оновлює адаптер для відображення нових повідомлень;
— прокручує список до останнього повідомлення.

Лістинг 2.38 реалізація методу onDataChange

```
chatReference.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
messagesArrayList.clear();
for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
msgModelclass messages = dataSnapshot.getValue(msgModelclass.class);
messagesArrayList.add(messages); }
mmessagesAdpter.notifyDataSetChanged();
messageAdpter.scrollToPosition(messagesArrayList.size() - 1); }
@Override
public void onCancelled(@NonNull DatabaseError error) {
}});
```

На останок залишилось реалізувати обробник подій натиснення кнопки відправлення повідомлення (ліст. 2.39).

Лістинг 2.39 обробник натиснення на кнопку надсилання повідомлень

```
sendbtn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
String message = textmsg.getText().toString();
if (message.isEmpty()) {
Toast.makeText(chatwindo.this, "Enter the message",
Toast.LENGTH_SHORT).show();
return;}
textmsg.setText("");
Date date = new Date();
msgModelclass messagess = new msgModelclass(message, SenderUID,
date.getTime());
DatabaseReference chatRef =
database.getReference().child("chats").child(senderRoom)
.child("messages").push();
String messageKey = chatRef.getKey();
messagess.setMessageKey(messageKey);
chatRef.setValue(messagess).addOnCompleteListener(new
OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (!task.isSuccessful()) {
Toast.makeText(chatwindo.this, "Failed to send message",
Toast.LENGTH_SHORT).show();
} else {
messageAdpter.scrollToPosition(messagesArrayList.size() - 1); }
database.getReference()
.child("chats")
.child(reciverRoom)
```

```
.child("messages")
.child(messageKey)
.setValue(messagess);
}});}});
```

Якщо поле вводу порожнє, відображає попередження. Далі отримуємо текст з поля вводу повідомлення та створюємо новий об'єкт повідомлення. Зберігаються повідомлення у Firebase Database в кімнаті відправника та одержувача. На цьому можна вважати реалізацію додатку закінченою.

# 3 ТЕСТУВАННЯ ФУНКЦІОНАЛЬНОСТІ ДОДАТКУ

Тестування мобільних додатків є важливим етапом в життєвому циклі розробки програмного забезпечення. З огляду на широкий спектр функціональності, різноманітність пристроїв і операційних систем, тестування мобільних додатків є складним і багатоаспектним процесом.

Перше, на що слід звернути увагу при тестуванні мобільних додатків, це сумісність. Існує безліч моделей пристроїв, які відрізняються розмірами екранів, потужністю процесорів, версіями операційних систем тощо. Це означає, що додаток, який працює бездоганно на одному пристрої, може мати проблеми на іншому. Тестувальники повинні переконатися, що додаток працює коректно на всіх цільових пристроях і версіях операційних систем. Сюди входить перевірка адаптивного інтерфейсу користувача, продуктивності додатка, його стабільності та швидкодії.

Важливо перевірити, чи не викликає додаток надмірне споживання батареї, що може призвести до швидкого розрядження пристрою. Крім того, тестувальники повинні оцінити час завантаження додатка, швидкість виконання основних функцій та його поведінку при роботі в умовах обмежених ресурсів.

Також необхідно приділити увагу тестуванню безпеки. Мобільні додатки часто мають доступ до конфіденційних даних користувачів, таких як особисті дані, фінансові інформація та інші чутливі дані. Тестувальники повинні переконатися, що додаток надійно захищає ці дані від несанкціонованого доступу та втрат. Це включає перевірку наявності шифрування даних, захист від атак типу «людина посередині», захист паролів та інших механізмів автентифікації.

Функціональність додатка також є важливим аспектом тестування. Вона включає в себе перевірку всіх функцій та можливостей додатка, щоб переконатися, що вони працюють відповідно до вимог і специфікацій. Це стосується як основних функцій, так і другорядних, а також інтеграції з іншими системами та сервісами.

Тестування мобільних додатків також включає в себе перевірку роботи додатка в різних умовах. Наприклад, тестувальники повинні оцінити поведінку додатка при втраті підключення до мережі, роботі в режимі слабкого сигналу, зміні орієнтації екрану, отриманні дзвінків під час роботи додатка тощо. Це дозволяє переконатися, що додаток стабільний і надійний в будь-яких умовах.

Смок-тестування, або smoke testing, є одним з важливих етапів тестування програмного забезпечення, включаючи мобільні додатки. Це тип поверхневого тестування, який виконується для того, щоб переконатися, що основні функції додатка працюють коректно. Смок-тестування зазвичай проводиться на початкових етапах тестування, щоб визначити, чи готове програмне забезпечення до подальшого, більш детального тестування.

Мета смок-тестування полягає в тому, щоб швидко виявити критичні помилки, які можуть унеможливити подальше тестування або використання додатка. Це дозволяє заощадити час і ресурси, оскільки виявлення таких помилок на ранніх етапах розробки допомагає запобігти їх розповсюдженню та ускладненню в майбутньому. Смок-тестування зазвичай охоплює основні функції та компоненти додатка, які є критично важливими для його роботи.

Прикладом смок-тестування може бути перевірка можливості запуску додатка, входу в систему, основної навігації між екранами, виконання базових операцій (наприклад, додавання або видалення елементів), а також перевірка основних функцій, таких як відправка повідомлень або здійснення покупок. Якщо під час смок-тестування виявляються критичні помилки, процес тестування може бути зупинений до їх виправлення.

Смок-тестування може бути автоматизованим або виконуватися вручну, в залежності від конкретних умов і вимог проекту. Автоматизація смок-тестування дозволяє швидко і ефективно проводити перевірку основних функцій після кожної зміни в коді, що забезпечує стабільність і надійність програмного забезпечення.

Щоб розпочати тестування необхідно інсталювати додаток на мобільний пристрій. Для цього потрібно отримати APK-пакет додатку. Це можна зробити у вкладці меню «Build» середовища Android Studio (рис. 3.1).

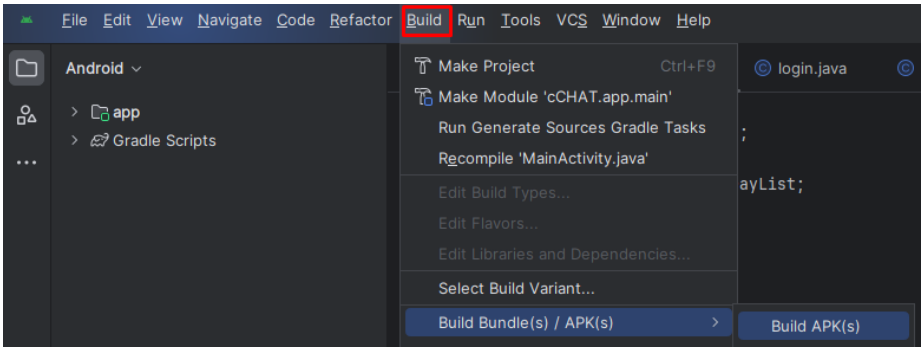


Рисунок 3.1 — Отримання APK-пакету додатку

Далі у випадному сповіщені, натиснувши на кнопку «locate», відчиняється вікно де знаходиться сам APK-пакет (рис. 3.2).


Рисунок 3.2 — Місцезнаходження APK-пакету

Після цього будь яким методом завантажуємо APK-пакет на мобільний пристрій та за допомогою нього інсталюємо додаток.

Інсталювавши додаток на мобільний пристрій можна переходити до самого тестування функціональності. Запустивши додаток нас зустрічає splash screen з анімацією елементів. Тривалість цієї активності дорівнює трьом секундам. Цього часу достатньо щоб прочитати всі надписи.

Після завершення роботи активності splash screen відчиняється активність входу. Це говорить про коректну роботу екрану запуску. Але спершу необхідно провести тестування активності реєстрації, так як за відсутності облікового запису немає можливості входу. Тому переходимо до активності реєстрації натиснувши на посилання «Sign up».

Для початку проведемо тестування вибору зображення профілю. Щоб це зробити необхідно натиснути на елемент з зображенням фотокамери. Після цього відкривається діалогове вікно в якому є можливість вибору зображення, яке розташоване у нас на смартфоні (рис. 3.3).

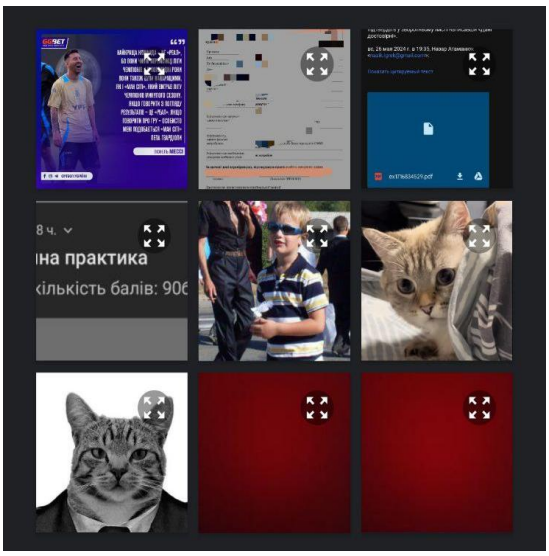

Рисунок 3.3 — Діалогове вікно для вибору зображення профілю

Після вибору зображення введемо некоректні дані, щоб протестувати чи відбувається валідація даних (рис. 3.4).

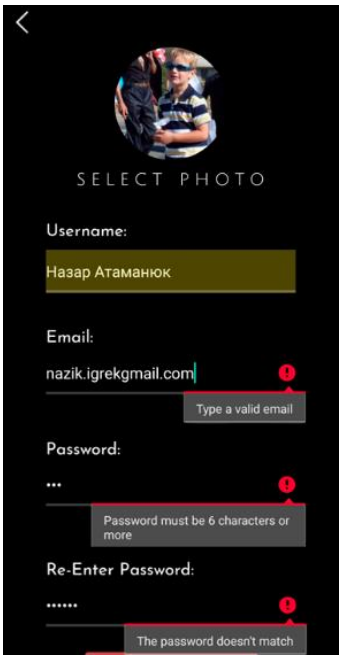


Рисунок 3.4 — Тестування валідації введених даних

Судячи з результату, валідність реалізована правильно, і все що нам залишається це перевірити реалізацію реєстрації облікового запису користувача при введенні коректних даних. Після реєстрації ми переходимо на головну активність. Також в базі даних реального часу з’явились відповідні записи про те що користувач створений, тому можна вважати що реєстрація відбулася успішно.

Щоб перевірити відображення користувачів на елементі RecyclerView, потрібно зареєструвати ще одного або декілька користувачів (рис. 3.5).

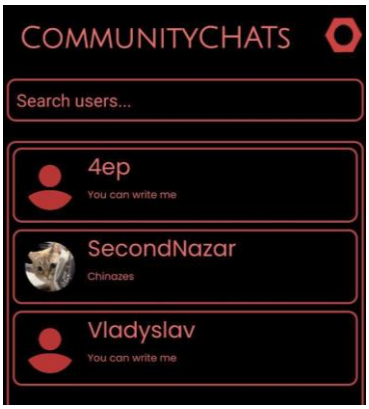


Рисунок 3.5 — Відображення користувачів

Відображення коректне, а отже переходимо до тестування активності налаштувань (рис. 3.6). Для цього натискаємо на відповідну кнопку в правому верхньому куті головної активності.

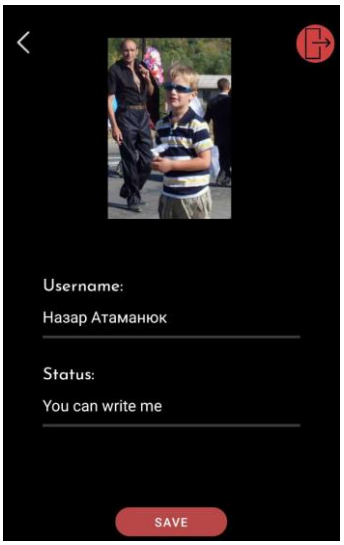


Рисунок 3.6 — Активність налаштувань

Спробуємо замінити всі дані, включаючи зображення профілю, та зберегти зміни натиснувши на кнопку «save». При повторному повернені до активності налаштувань після внесення змін, ми бачимо що дані були успішно збережені (рис. 3.7).


Рисунок 3.7 — Збережені зміни інформації профілю

Далі протестуємо можливість виходу з облікового запису. Для цього потрібно натиснути на відповідну кнопку, яка знаходиться на активності налаштувань, у верхньому правому куті та в діалоговому вікні натиснути «yes» (рис. 3.8).

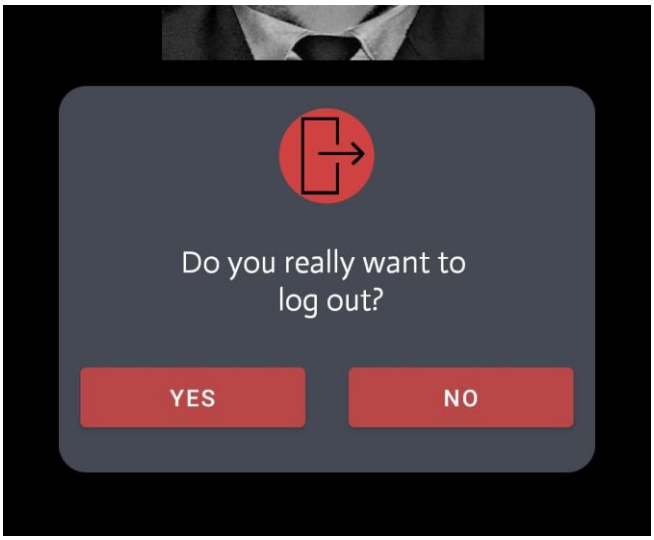


Рисунок 3.8 — Діалогове вікно для виходу з облікового запису

Після підтвердження ми потрапляємо на активність входу, а отже все працює коректно.

Маючи зареєстрований обліковий запис, тепер є можливість протестувати активність входу. Тому вводимо некоректні дані у відповідні поля. Для початку перевіримо реакцію на введення незареєстрованої електронної пошти (рис. 3.9).

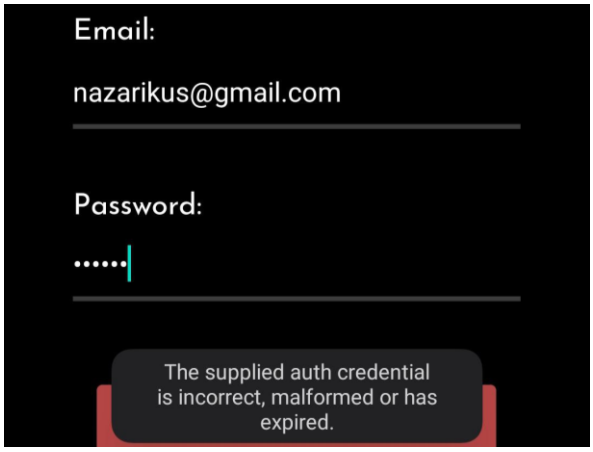


Рисунок 3.9 — Реакція на авторизацію з незареєстрованою електронною поштою

Тепер протестуємо валідацію даних при введенні некоректної електронної пошти та паролю що менший за 6 символів (рис. 3.10).

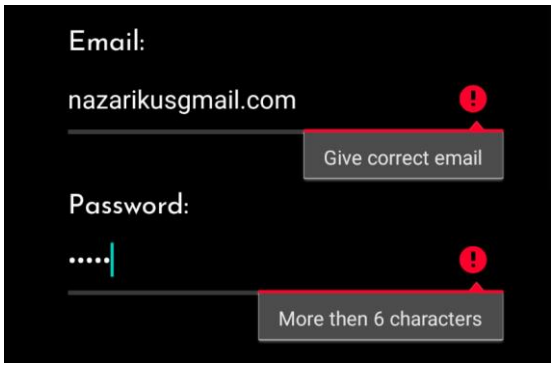


Рисунок 3.10 — Тестування валідації даних при спробі входу в обліковий запис

Далі відбувається тестування входу в обліковий запис при введенні коректних даних уже зареєстрованого облікового запису. Так як ми потрапляємо в головну активність що зображена на рисунку 3.5, то можна стверджувати що активність входу працює правильно.

Тепер протестуємо поле пошуку користувачів. Для цього спершу введемо в поле символи, які не мають збігів з іменем користувачів, і після цього введемо символи з збігами. Результат обох варіантів зображений на рисунку 3.11.

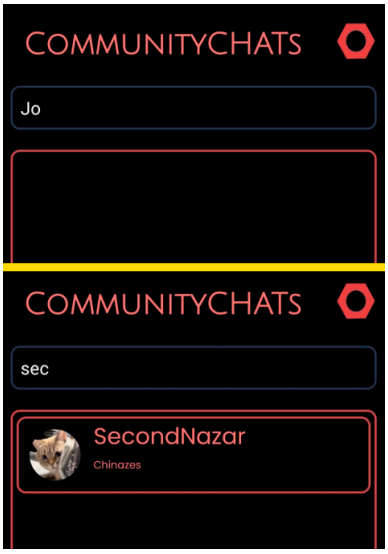


Рисунок 3.11 — Тестування пошукового поля

Залишилось перевірити активність чату та можливість листування текстовими повідомленнями між користувачами. Тому в головній активності натискаємо на будь-якого з користувачів, після чого відбувається перехід до активності чату. Тут ми можемо перевірити можливість обміну повідомленнями, тому надсилаємо повідомлення та чекаємо на відповідь (рис. 3.12).

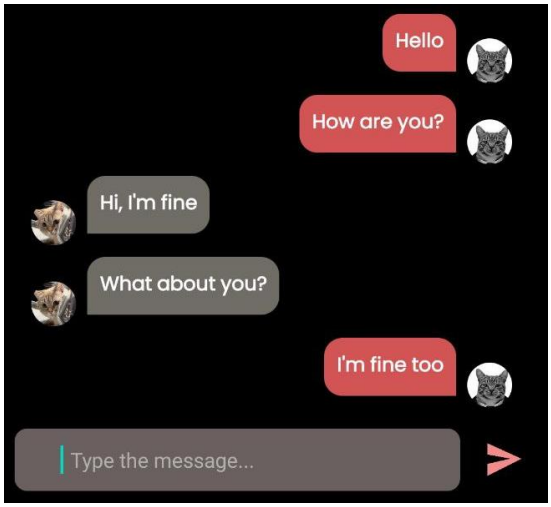


Рисунок 3.12 — Тестування обміну текстовими повідомленнями

На останок проведемо тестування можливості видалення повідомлень. Для цього необхідно затримати натиснення на повідомлення, після чого з’являється діалогове вікно для підтвердження операції (рис. 3.13).

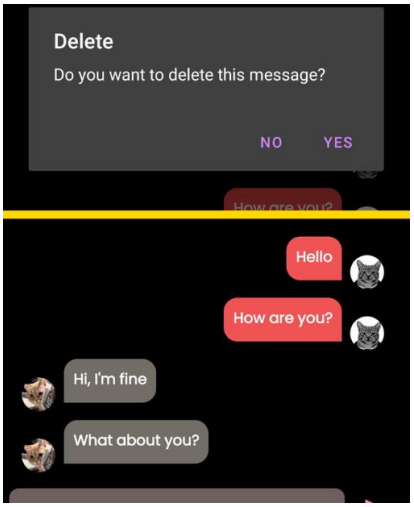


Рисунок 3.13 — Тестування можливості видалення повідомлень

Як показано на рисунку, видалення повідомлень дійсно працює. Це говорить про те що додаток повністю пройшов тестування та вважається готовим для використання.

У підсумку, смок-тестування є невід'ємною частиною процесу тестування мобільних додатків. Воно дозволяє швидко виявляти критичні проблеми на ранніх етапах розробки, забезпечуючи стабільність і якість програмного забезпечення. Смок-тестування допомагає зекономити час і ресурси, забезпечуючи ефективність подальшого тестування та розробки.

# ВИСНОВКИ

У результаті проведеної роботи було розроблено повноцінний мобільний додаток для оперативної комунікації, що функціонує на платформі Android. Процес розробки був реалізований у середовищі Android Studio, що дозволило максимально ефективно використовувати інструменти для створення сучасних і функціональних додатків. Основним напрямом інтеграції додатка стала платформа Firebase, яка забезпечила низку ключових функцій для успішної реалізації проекту.

Зокрема, було використано Firebase Authentication для управління автентифікацією користувачів, що дозволило забезпечити безпечний доступ до додатка та управління користувацькими сесіями. Інтеграція з Firebase Realtime Database забезпечила швидкий і надійний обмін даними в реальному часі, що є критично важливим для додатків цього типу. Крім того, використання Firebase Storage надало можливість ефективно зберігати і обробляти мультимедійні файли, такі як зображення.

Розробка додатка включала шість активностей, кожна з яких відповідає за різні аспекти функціональності. Це дозволило створити інтуїтивно зрозумілий і зручний для користувачів інтерфейс, що полегшує комунікацію та взаємодію з додатком. Важливо відзначити, що проект передбачає високу масштабованість і можливість подальшого розвитку функціоналу, що відкриває перспективи для його адаптації під різні потреби користувачів.

Отже, розроблений додаток для оперативної комунікації на платформі Android, завдяки інтеграції з Firebase, продемонстрував високу ефективність і зручність використання. Робота над проектом дозволила отримати цінний досвід в області мобільної розробки та роботи з хмарними сервісами, а також заклала основу для подальшого вдосконалення та розширення функціональності додатка. Дослідження та розробка в рамках цієї роботи стали важливим кроком у підвищенні рівня кваліфікації у сфері інформаційних технологій та забезпеченні інноваційних підходів до розв'язання задач оперативної комунікації.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ


1. AR and VR Impact on Mobile App Development: вебсайт. URL: https://attractgroup.com/blog/the-impact-of-ar-and-vr-on-app-development/ (дата звернення: 03.03.2024).
2. 6 Business Benefits of Mobile Communication Technology: вебсайт. URL: https://www.leapxpert.com/6-business-benefits-of-mobile-communication-technology/ (дата звернення: 03.03.2024).
3. Data Encryption Security in Mobile and Cloud Computing Environments: вебсайт. URL: https://www.researchgate.net/publication/340397744_Data_Encryption_Security_in_Mobile_and_Cloud_Computing_Environments (дата звернення: 03.03.2024).
4. On Fundamental Concept Of Mobility For Mobile Communications: вебсайт. URL: https://www.researchgate.net/publication/2546738_On_Fundamental_Concept_Of_Mobility_For_Mobile_Communications (дата звернення: 05.03.2024).
5. Cross-Platform Application Development: вебсайт. URL: https://itechcraft.com/technologies/cross-platform/ (дата звернення: 09.03.2024).
6. Mobile App Development for iOS and Android : вебсайт. URL: https://neklo.com/blog/ios-vs-android-development (дата звернення: 15.03.2024).
7. Mobile Operating System Market Share Worldwide : вебсайт. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення: 15.03.2024).
8. Best Gmail Integrations to Increase Your Email : вебсайт. URL: https://www.smartsuite.com/blog/gmail-integrations-productivity (дата звернення: 17.03.2024).

9. Android Studio: Google's Official IDE for Android App Development: вебсайт. URL: https://www.linkedin.com/pulse/android-studio-googles-official-ide-app-development-flmoe (дата звернення: 17.03.2024).
10. Design for Android: вебсайт. URL: https://developer.android.com/design/ui (дата звернення: 17.03.2024).
11. Firebase Вікіпедія: вебсайт. URL: https://uk.wikipedia.org/wiki/Firebase (дата звернення: 07.04.2024).
12. Firebase's policies for introducing changes and communicating those changes : вебсайт. URL: https://firebase.google.com/policies/changes-to-firebase/introducing-and-communicating-changes (дата звернення: 08.04.2024).
13. Collect feedback from testers: вебсайт. URL: https://firebase.google.com/docs/app-distribution/collect-feedback-from-testers (дата звернення: 08.04.2024).
14. Splash Screen: Why Your App Needs One & Crucial Points To Follow: вебсайт. URL: https://www.cleveroad.com/blog/mobile-app-splash-screen-reasons-to-integrate-launch-screen-into-your-app/ (дата звернення: 12.04.2024).
15. What is Login Authentication: вебсайт. URL: https://www.loginradius.com/blog/identity/what-is-login-authentication/ (дата звернення: 15.04.2024).
16. Атаманюк Н. Р. Використання Firebase для розробки мережного програмного забезпечення для оперативної комунікації на платформі Android Міжнародна науково-методична Інтернет-конференція Проблеми вищої математичної освіти, виклики сучасності: тези доповідей. URL: https://conferences.vntu.edu.ua/index.php/pmovc/pmovc24/schedConf/presentations

# ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
« 6 » травня 2024 року

ТЕХНІЧНЕ ЗАВДАННЯ
На виконання бакалаврської дипломної роботи
«Мережне програмне забезпечення для оперативної комунікації на платформі Android»
08-54.БДР.016.00.000.ПЗ

Науковий керівник: к.т.н., проф. каф. ОТ
_____ Захарченко С.М.
Студент групи 2СП-20б
_____ Атаманюк Н.Р.

ВНТУ 2024

1 Підстава для виконання бакалаврської дипломної роботи (БДР)

1.1 Оперативна комунікація є важливим елементом зв’язку, реалізація якої вимагає значних зусиль для розробників. З розвитком інформаційних технологій виникла потреба у створенні ефективних методів розробки комунікаційних систем. Одним із таких інструментів є мобільні додатки, які дозволяють зручно організувати комунікаційні процеси.

1.2 Наказ про затвердження теми БДР від «11» березня 2024 року № 80.

2 Мета БДР і призначення розробки

2.1 Мета роботи – розробка мережного програмного забезпечення для оперативної комунікації на платформі Android, яке дозволить зручно та ефективно обмінюватись інформацією між користувачами.

2.2 Призначення розробки – реалізація зручного мобільного додатку, який забезпечить просту систему авторизації, ефективний пошук користувачів, налаштування інформації про користувача та обмін текстовими повідомленнями.

3 Вихідні дані для використання БДР

3.1 Огляд сучасних додатків для оперативної комунікації на платформі Android.

3.2 Аналіз методів розробки.

3.3 Розробка додатку для оперативної комунікації на платформі Android.

3.4 Тестування додатку для оперативної комунікації на платформі Android.

4 Вимоги до виконання БДР

Головна вимога – програмний засіб має бути реалізований на платформі Android та повинен надавати можливість обміну текстовими повідомленнями між користувачами.

5 Етапи БДР та очікувані результати

Етапи роботи та очікувані результати наведено у Таблиці А.1.

Таблиця А.1 – Етапи БДР

| № | Назва та зміст етапу | Термін виконання | Примітка |
|---|---|---|---|
| 1. | Постановка задачі роботи | 1.03.24 | |
| 2. | Огляд сучасних додатків для оперативної комунікації на платформі Android | 02.03-25.03.24 | Розділ 1 |
| 3. | Аналіз потреб користувачів | 26.03-05.04.24 | Розділ 1 |
| 4. | Аналіз методів розробки | 06.04-10.04.24 | Розділ 1 |
| 5. | Розробка додатку для оперативної комунікації на платформі Android | 11.04-30.04.24 | Розділ 2 |
| 6. | Тестування додатку для оперативної комунікації на платформі Android | 01.05-06.05.24 | Розділ 3 |
| 7. | Оформлення пояснювальної записки | 07.05-30.05.24 | |
| 8. | Перевірка якості виконання бакалаврської роботи | 31.05-10.06.24 | |

6 Матеріали, що подаються до захисту БДР

До захисту подаються: пояснювальна записка БДР, ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника, анотації до БДР українською та іноземною мовами, довідка про відповідність оформлення БДР діючим вимогам.

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист БДР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлення та порядок виконання БДР

8.1 При оформленні БКР використовується:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп'ютерна інженерія» (освітня програма «Системне програмування»). Кафедра обчислювальної техніки ВНТУ 2023;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

# ДОДАТОК Б

Структурна схема

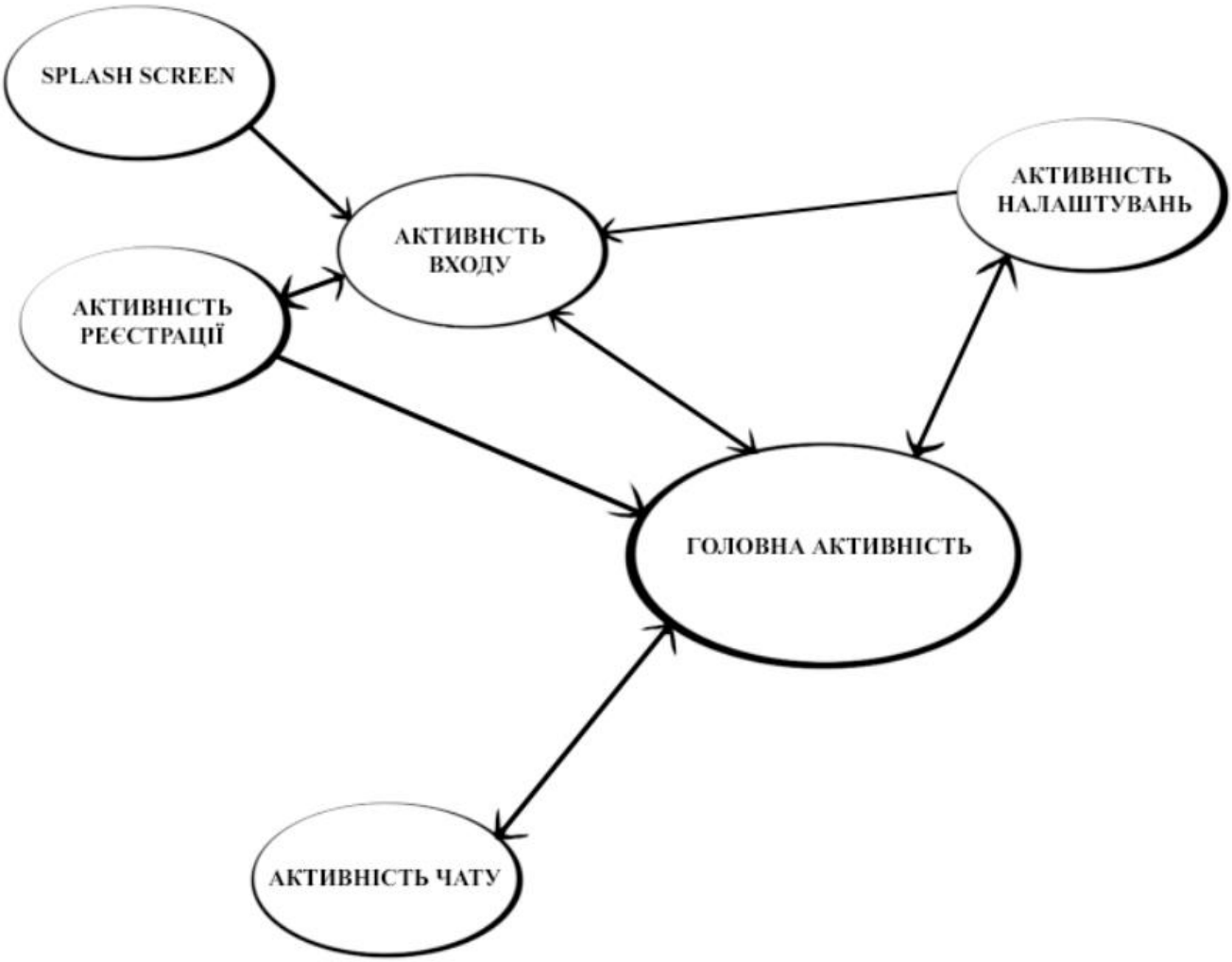


Рисунок А.1 — Структурна схема активностей додатку

# ДОДАТОК В

Лістинг файлу splash.java

```
package com.c.CHAT;
import androidx.appcompat.app.AppCompatActivity;
import android.annotation.SuppressLint;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;
import android.widget.TextView;
import com.google.firebase.auth.FirebaseAuth;
public class splash extends AppCompatActivity {
ImageView logo;
TextView name, own1, own2;
Animation topAnim, bottomAnim;
@SuppressLint("MissingInflatedId")
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_splash);
getSupportActionBar().hide();
logo = findViewById(R.id.logoimg);
name = findViewById(R.id.logonameimg);
own1 = findViewById(R.id.ownone);
own2 = findViewById(R.id.owntwo);
topAnim = AnimationUtils.loadAnimation(this, R.anim.top_animation);
bottomAnim = AnimationUtils.loadAnimation(this, R.anim.bottom_animation);
logo.setAnimation(topAnim);
name.setAnimation(bottomAnim);
own1.setAnimation(bottomAnim);
own2.setAnimation(bottomAnim);
new Handler().postDelayed(new Runnable() {
```

```
@Override
public void run() {
FirebaseAuth auth = FirebaseAuth.getInstance();
Intent intent;
if (auth.getCurrentUser() != null) {
intent = new Intent(splash.this, MainActivity.class);
} else {
intent = new Intent(splash.this, login.class);
}
startActivity(intent);
finish();
}}, 3000);}}
```

# ДОДАТОК Г

Лістинг файлу login.java

```
package com.c.CHAT;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
public class login extends AppCompatActivity {
TextView logsignup;
Button button;
EditText email, password;
FirebaseAuth auth;
String emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+";
android.app.ProgressDialog progressDialog;
@Override
public void onBackPressed() {
super.onBackPressed();
finishAffinity();}
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_login);
getSupportActionBar().hide();
```

```
progressDialog = new ProgressDialog(this);
progressDialog.setMessage("Please wait...");
progressDialog.setCancelable(false);
auth = FirebaseAuth.getInstance();
button = findViewById(R.id.logbutton);
email = findViewById(R.id.editTexLogEmail);
password = findViewById(R.id.editTextLogPassword);
logsignup = findViewById(R.id.logsignup);
logsignup.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(login.this,registration.class);
startActivity(intent);
finish();}});
button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
String Email = email.getText().toString();
String pass = password.getText().toString();
if (TextUtils.isEmpty(Email) && TextUtils.isEmpty(pass)){
progressDialog.dismiss();
Toast.makeText(login.this, "Enter the data", Toast.LENGTH_SHORT).show();
}else if (TextUtils.isEmpty(pass)){
progressDialog.dismiss();
Toast.makeText(login.this, "Enter the password", Toast.LENGTH_SHORT).show();
}else if (TextUtils.isEmpty(Email)) {
progressDialog.dismiss();
Toast.makeText(login.this, "Enter the email", Toast.LENGTH_SHORT).show();
}else if (!Email.matches(emailPattern)){
progressDialog.dismiss();
email.setError("Give correct email");
}else if (password.length()<6){
progressDialog.dismiss();
password.setError("More then 6 characters");
}else {
```

```
auth.signInWithEmailAndPassword(Email,pass).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
@Override
public void onComplete(@NonNull Task<AuthResult> task) {
if (task.isSuccessful()){
progressDialog.show();
try {
Intent intent = new Intent(login.this , MainActivity.class);
startActivity(intent);
finish();
}catch (Exception e){
Toast.makeText(login.this, e.getMessage(), Toast.LENGTH_SHORT).show();
}
}else {
Toast.makeText(login.this, task.getException().getMessage(), Toast.LENGTH_SHORT).show();
}}});}}});}}
```

# ДОДАТОК Д

Лістинг файлу registration.java

```
package com.c.CHAT;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;
import de.hdodenhof.circleimageview.CircleImageView;
public class registration extends AppCompatActivity {
ImageView loginbut;
EditText rg_username, rg_email , rg_password, rg_repassword;
Button rg_signup;
CircleImageView rg_profileImg;
FirebaseAuth auth;
Uri imageURI;
```

```
String imageuri;
String emailPattern = "[a-zA-Z0-9._-]+@[a-z]+\\.+[a-z]+";
FirebaseDatabase database;
FirebaseStorage storage;
ProgressDialog progressDialog;
@Override
public void onBackPressed() {
super.onBackPressed();
finishAffinity();}
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_registration);
getSupportActionBar().hide();
progressDialog = new ProgressDialog(this);
progressDialog.setMessage("Establishing the account");
progressDialog.setCancelable(false);
database = FirebaseDatabase.getInstance();
storage = FirebaseStorage.getInstance();
auth = FirebaseAuth.getInstance();
loginbut = findViewById(R.id.loginbut);
rg_username = findViewById(R.id.rgusername);
rg_email = findViewById(R.id.rgemail);
rg_password = findViewById(R.id.rgpassword);
rg_repassword = findViewById(R.id.rgrepassword);
rg_profileImg = findViewById(R.id.profilerg0);
rg_signup = findViewById(R.id.signupbutton);
loginbut.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(registration.this,login.class);
startActivity(intent);
finish();}});
rg_signup.setOnClickListener(new View.OnClickListener() {
@Override
```

```
public void onClick(View v) {
String namee = rg_username.getText().toString();
String emaill = rg_email.getText().toString();
String Password = rg_password.getText().toString();
String cPassword = rg_repassword.getText().toString();
String status = "You can write me";
if (TextUtils.isEmpty(namee) || TextUtils.isEmpty(emaill) ||
TextUtils.isEmpty(Password) || TextUtils.isEmpty(cPassword)){
progressDialog.dismiss();
Toast.makeText(registration.this, "Please enter valid information", Toast.LENGTH_SHORT).show();
}else  if (!emaill.matches(emailPattern)){
progressDialog.dismiss();
rg_email.setError("Type a valid email");
}else if (Password.length()<6){
progressDialog.dismiss();
rg_password.setError("Password must be 6 characters or more");
}else if (!Password.equals(cPassword)){
progressDialog.dismiss();
rg_repassword.setError("The password doesn't match");
}else {
auth.createUserWithEmailAndPassword(emaill,Password).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
@Override
public void onComplete(@NonNull Task<AuthResult> task) {
if (task.isSuccessful()){
String id = task.getResult().getUser().getUid();
DatabaseReference reference = database.getReference().child("user").child(id);
StorageReference storageReference = storage.getReference().child("Upload").child(id);
if (imageURI!=null){
storageReference.putFile(imageURI).addOnCompleteListener(new
OnCompleteListener<UploadTask.TaskSnapshot>() {
@Override
public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {
if (task.isSuccessful()){
storageReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
```

```
@Override
public void onSuccess(Uri uri) {
imageuri = uri.toString();
Users users = new Users(id,namee,emaill,Password,imageuri,status);
reference.setValue(users).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()){
progressDialog.show();
Intent intent = new Intent(registration.this,MainActivity.class);
startActivity(intent);
finish();
}else {
Toast.makeText(registration.this, "Error in creating the user", Toast.LENGTH_SHORT).show();
}}});}});}});
}else {
imageuri = "https://firebasestorage.googleapis.com/v0/b/cchat-
e014d.appspot.com/o/defaultavatar.png?alt=media&token=37818586-2538-4597-926c-
008a915562f8";
Users users = new Users(id,namee,emaill,Password,imageuri,status);
reference.setValue(users).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()){
progressDialog.show();
Intent intent = new Intent(registration.this,MainActivity.class);
startActivity(intent);
finish();
}else {
Toast.makeText(registration.this, "Error in creating the user", Toast.LENGTH_SHORT).show();
}});}
}else {
Toast.makeText(registration.this, task.getException().getMessage(),
Toast.LENGTH_SHORT).show();
}});}});
```

```
rg_profileImg.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent();
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent,"Select Picture"),10);
}});}
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
super.onActivityResult(requestCode, resultCode, data);
if (requestCode==10){
if (data!=null){
imageURI = data.getData();
rg_profileImg.setImageURI(imageURI);
}}}}
```

# ДОДАТОК Е

## Лістинг файлу MainActivity.java

```
package com.c.CHAT;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.annotation.SuppressLint;
import android.content.Intent;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import java.util.ArrayList;
public class MainActivity extends AppCompatActivity {
FirebaseAuth auth;
RecyclerView mainUserRecyclerView;
UserAdpter adapter;
FirebaseDatabase database;
ArrayList<Users> usersArrayList;
ImageView setbut;
EditText searchBar;
@Override
public void onBackPressed() {
super.onBackPressed();
finishAffinity();}
```

```
@SuppressLint("MissingInflatedId")
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
getSupportActionBar().hide();
database = FirebaseDatabase.getInstance();
auth = FirebaseAuth.getInstance();
setbut = findViewById(R.id.settingBut);
searchBar = findViewById(R.id.searchBar);
DatabaseReference reference = database.getReference().child("user");
usersArrayList = new ArrayList<>();
mainUserRecyclerView = findViewById(R.id.mainUserRecyclerView);
mainUserRecyclerView.setLayoutManager(new LinearLayoutManager(this));
adapter = new UserAdpter(MainActivity.this, usersArrayList);
mainUserRecyclerView.setAdapter(adapter);
reference.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
usersArrayList.clear();
String currentUserId = auth.getCurrentUser().getUid();
for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
Users users = dataSnapshot.getValue(Users.class);
if (!users.getUserId().equals(currentUserId)) {
usersArrayList.add(users); }}
adapter.setUsersList(usersArrayList);
adapter.notifyDataSetChanged();}
@Override
public void onCancelled(@NonNull DatabaseError error) {
}});
setbut.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(MainActivity.this, setting.class);
startActivity(intent); }});
```

```
searchBar.addTextChangedListener(new TextWatcher() {
@Override
public void beforeTextChanged(CharSequence s, int start, int count, int after) {
}
@Override
public void onTextChanged(CharSequence s, int start, int before, int count) {
adapter.filter(s.toString());}
@Override
public void afterTextChanged(Editable s) {
}});}}
```

# ДОДАТОК Ж

Лістинг файлу chatwindo.java

```
package com.c.CHAT;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.cardview.widget.CardView;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.squareup.picasso.Picasso;
import java.util.ArrayList;
import java.util.Date;
import de.hdodenhof.circleimageview.CircleImageView;
public class chatwindo extends AppCompatActivity {
String reciverimg, reciverUid, reciverName, SenderUID;
CircleImageView profile;
TextView reciverNName;
FirebaseDatabase database;
FirebaseAuth firebaseAuth;
public static String senderImg;
```

```
public static String reciverIImg;
CardView sendbtn;
EditText textmsg;
ImageView mainbutback;
static String senderRoom;
static String reciverRoom;
RecyclerView messageAdpter;
ArrayList<msgModelclass> messagesArrayList;
messagesAdpter mmessagesAdpter;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_chatwindo);
getSupportActionBar().hide();
database = FirebaseDatabase.getInstance();
firebaseAuth = FirebaseAuth.getInstance();
reciverName = getIntent().getStringExtra("nameeee");
reciverimg = getIntent().getStringExtra("reciverImg");
reciverUid = getIntent().getStringExtra("uid");
messagesArrayList = new ArrayList<>();
sendbtn = findViewById(R.id.sendbtnn);
textmsg = findViewById(R.id.textmsg);
reciverNName = findViewById(R.id.recivername);
profile = findViewById(R.id.profileimgg);
messageAdpter = findViewById(R.id.msgadpter);
LinearLayoutManager linearLayoutManager = new LinearLayoutManager(this);
linearLayoutManager.setStackFromEnd(true);
messageAdpter.setLayoutManager(linearLayoutManager);
mmessagesAdpter = new messagesAdpter(chatwindo.this, messagesArrayList);
messageAdpter.setAdapter(mmessagesAdpter);
Picasso.get().load(reciverimg).into(profile);
reciverNName.setText(reciverName);
SenderUID = firebaseAuth.getUid();
senderRoom = SenderUID + reciverUid;
reciverRoom = reciverUid + SenderUID;
```

```
DatabaseReference reference = database.getReference().child("user").child(firebaseAuth.getUid());
DatabaseReference chatReference =
database.getReference().child("chats").child(senderRoom).child("messages");
chatReference.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
messagesArrayList.clear();
for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
msgModelclass messages = dataSnapshot.getValue(msgModelclass.class);
messagesArrayList.add(messages);}
mmessagesAdpter.notifyDataSetChanged();
messageAdpter.scrollToPosition(messagesArrayList.size() - 1);}
@Override
public void onCancelled(@NonNull DatabaseError error) {
}});
reference.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
senderImg = snapshot.child("profilepic").getValue().toString();
reciverIImg = reciverimg;}
@Override
public void onCancelled(@NonNull DatabaseError error) {
}});
sendbtn.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
String message = textmsg.getText().toString();
if (message.isEmpty()) {
Toast.makeText(chatwindo.this, "Enter the message", Toast.LENGTH_SHORT).show();
return; }
textmsg.setText("");
Date date = new Date();
msgModelclass messagess = new msgModelclass(message, SenderUID, date.getTime());
DatabaseReference chatRef =
database.getReference().child("chats").child(senderRoom).child("messages").push();
```

```
String messageKey = chatRef.getKey();
messagess.setMessageKey(messageKey);
chatRef.setValue(messagess).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (!task.isSuccessful()) {
Toast.makeText(chatwindo.this, "Failed to send message", Toast.LENGTH_SHORT).show();
} else {
messageAdpter.scrollToPosition(messagesArrayList.size() - 1);}
database.getReference()
.child("chats")
.child(reciverRoom)
.child("messages")
.child(messageKey)
.setValue(messagess); }});}});
mainbutback = findViewById(R.id.mainbutback);
mainbutback.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
startActivity(new Intent(chatwindo.this, MainActivity.class));
finish();
}});}}
```

# ДОДАТОК И

Лістинг файлу setting.java

```
package com.c.CHAT;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.storage.FirebaseStorage;
import com.google.firebase.storage.StorageReference;
import com.google.firebase.storage.UploadTask;
import com.squareup.picasso.Picasso;
public class setting extends AppCompatActivity {
ImageView setprofile;
EditText setname, setstatus;
Button donebut;
FirebaseAuth auth;
```

```
FirebaseDatabase database;
FirebaseStorage storage;
Uri setImageUri;
String email,password;
ProgressDialog progressDialog;
ImageView imglogout, mainbutback;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_setting);
getSupportActionBar().hide();
auth = FirebaseAuth.getInstance();
database = FirebaseDatabase.getInstance();
storage = FirebaseStorage.getInstance();
setprofile = findViewById(R.id.settingprofile);
setname = findViewById(R.id.settingname);
setstatus = findViewById(R.id.settingstatus);
donebut = findViewById(R.id.donebutt);
progressDialog = new ProgressDialog(this);
progressDialog.setMessage("Saving...");
progressDialog.setCancelable(false);
mainbutback = findViewById(R.id.mainbutback);
mainbutback.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(setting.this,MainActivity.class);
startActivity(intent);
finish();
}});
imglogout = findViewById(R.id.logoutimg);
imglogout.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Dialog dialog = new Dialog(setting.this,R.style.dialoge);
dialog.setContentView(R.layout.dialog_layout);
```

```
Button no,yes;
yes = dialog.findViewById(R.id.yesbnt);
no = dialog.findViewById(R.id.nobnt);
yes.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
FirebaseAuth.getInstance().signOut();
Intent intent = new Intent(setting.this,login.class);
startActivity(intent);
finish();
}});
no.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
dialog.dismiss();
}});
dialog.show();
}});
DatabaseReference reference = database.getReference().child("user").child(auth.getUid());
StorageReference storageReference = storage.getReference().child("Upload").child(auth.getUid());
reference.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
email = snapshot.child("mail").getValue().toString();
password = snapshot.child("password").getValue().toString();
String name = snapshot.child("userName").getValue().toString();
String profile = snapshot.child("profilepic").getValue().toString();
String status = snapshot.child("status").getValue().toString();
setname.setText(name);
setstatus.setText(status);
Picasso.get().load(profile).into(setprofile); }
@Override
public void onCancelled(@NonNull DatabaseError error) {
}});
setprofile.setOnClickListener(new View.OnClickListener() {
```

```
@Override
public void onClick(View view) {
Intent intent = new Intent();
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent, "Select Picture"), 10);
}});
donebut.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
progressDialog.show();
String name = setname.getText().toString();
String Status = setstatus.getText().toString();
if (setImageUri!=null){
storageReference.putFile(setImageUri).addOnCompleteListener(new
OnCompleteListener<UploadTask.TaskSnapshot>() {
@Override
public void onComplete(@NonNull Task<UploadTask.TaskSnapshot> task) {
storageReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
@Override
public void onSuccess(Uri uri) {
String finalImageUri = uri.toString();
Users users = new Users(auth.getUid(), name,email,password,finalImageUri,Status);
reference.setValue(users).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()){
progressDialog.dismiss();
Toast.makeText(setting.this, "Data Is save ", Toast.LENGTH_SHORT).show();
Intent intent = new Intent(setting.this,MainActivity.class);
startActivity(intent);
finish();
}else {
progressDialog.dismiss();
Toast.makeText(setting.this, "error", Toast.LENGTH_SHORT).show();
```

```
}}}});}});}});
}else {
storageReference.getDownloadUrl().addOnSuccessListener(new OnSuccessListener<Uri>() {
@Override
public void onSuccess(Uri uri) {
String finalImageUri = uri.toString();
Users users = new Users(auth.getUid(), name,email,password,finalImageUri,Status);
reference.setValue(users).addOnCompleteListener(new OnCompleteListener<Void>() {
@Override
public void onComplete(@NonNull Task<Void> task) {
if (task.isSuccessful()){
progressDialog.dismiss();
Toast.makeText(setting.this, "Data Is save ", Toast.LENGTH_SHORT).show();
Intent intent = new Intent(setting.this,MainActivity.class);
startActivity(intent);
finish();
}else {
progressDialog.dismiss();
Toast.makeText(setting.this, "Mistake", Toast.LENGTH_SHORT).show();
}}});}});}}});}
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
super.onActivityResult(requestCode, resultCode, data);
if (requestCode == 10) {
if (data != null) {
setImageUri = data.getData();
setprofile.setImageURI(setImageUri);
}}}}
```

# ДОДАТОК К

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Мережне програмне забезпечення для оперативної комунікації на платформі Android

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: кафедра обчислювальної техніки
(кафедра, факультет)

**Показники звіту подібності Unicheck**

Оригінальність 98,7% Схожість 1,3%

Аналіз звіту подібності (відмітити потрібне):

□ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

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

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

Особа, відповідальна за перевірку _____ Захарченко С. М.
*(підпис)* *(прізвище, ініціали)*

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Атаманюк Н.Р.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Захарченко С.М.
(підпис) (прізвище, ініціали)