КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

122 «Комп'ютерні науки»
(шифр і назва спеціальності)

«Прикладне програмування»
(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Мобільний застосунок месенджер»

Виконав
Лаговський Ярослав Вікторович
(прізвище, ім'я, по батькові)

Керівник Гарко Ірина Ігорівна
(прізвище, ім'я, по батькові)

Попередній захист:

_____
_
(Висновок: "До захисту в екзаменаційній комісії ")

Завідувач кафедри Плескач В.Л. _____

Засвідчую, що у цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань. Унікальність тексту роботи - 90%

Київ – 2022

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	09.10.2021	
2.	Видача завдання кваліфікаційної роботи бакалавра	19.10.2021	
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	21.10.2021	
4.	Затвердження плану кваліфікаційної роботи бакалавра	25.10.2022	
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	01.11.2022	
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	21.12.2022	
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	31.01.2022	
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	30.03.2022	
9.	Подання роботи у першому варіанті	28.04.2022	
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2022	
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	23.05.2022	
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	27.05.2022	
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	10.06.2022	
14.	Захист кваліфікаційної роботи бакалавра	22.06.2022 23.06.2022 24.06.2022	

Здобувач вищої освіти ____ _____

Керівник _____ _____

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини кваліфікаційної роботи	Обсяг, арк.
Титульний аркуш	1
Календарний план кваліфікаційної роботи	1
Відомість кваліфікаційної роботи	1
Анотація	1
Анотація (іноземною мовою-англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	0
Вступ	2
Розділ 1	10
Розділ 2	10
Розділ 3	37
Висновки	1
Список використаних джерел	2
Додатки	87

<table>
<tr><td></td><td></td><td></td><td></td><td colspan="3" rowspan="2">ДП ХХХХ 00.000.00</td></tr>
<tr><td></td><td>ПІБ</td><td>Підп.</td><td>Дата</td></tr>
<tr><td>Розробн.</td><td>Лаговський Я.В.</td><td></td><td></td><td rowspan="4">Відомість кваліфікаційної роботи</td><td>Лист</td><td>Листів</td></tr>
<tr><td>Керівн.</td><td>Гарко І.І.</td><td></td><td></td><td></td><td></td></tr>
<tr><td>Н/контр.</td><td>Базилюк А.М.</td><td></td><td rowspan="2"></td></tr>
<tr><td>Зав.каф.</td><td>Плескач В.Л.</td><td></td></tr>
</table>

АНОТАЦІЯ

Кваліфікаційна робота бакалавра містить: 156 сторінок, 54 рисунки, 15 використаних джерел. Метою роботи є підвищення зручності спілкування між людьми з використанням розробленого мобільного застосунку. Об’єктом дослідження є листування в групах і приватних чатах з використанням Android застосунку. Предмет дослідження – програмні підходи до побудови рішень для заповнення інформації, що зберігається в базі даних, особливості та основні принципи розроблення користувацьких інтерфейсів. Методи дослідження - системний аналіз і синтез, вибір та застосування методу розробки мобільних застосунків, порівняння функціоналу IDE для конструювання Android застосунку. У процесі виконання кваліфікаційної роботи бакалавра було використано: програмне забезпечення Android studio, документація Kotlin, XML міжнародні та національні стандарти з побудови програмних систем, стандарти з побудов Android застосунків та користувацьких інтерфейсів.

В результаті виконання кваліфікаційної роботи бакалавра було досягнуто мету та отримано такі результати:

Було створено застосунок, що реалізує такі функції:

- додавання контактів, які користуються застосунком, до списку контактів;
- приватні чати;
- групові чати;
- можливість відправлення файлів, картинок, голосових повідомлень.

Програмна реалізація виконана мовою програмування Kotlin.

Ключові слова: Android застосунок, мессенджер, firebase, Kotlin.

ABSTRACT

The bachelor's thesis contains: 156 pages, 54 drawings, 15 sources used. The aim of the work is to increase the convenience of communication between people using the developed mobile application. The object of the study is correspondence in groups and private chats using the Android application. The subject of the research is software approaches to building solutions for filling in the information stored in the database, features and basic principles of user interface development. Research methods - system analysis and synthesis, selection and application of the method of mobile application development, comparison of IDE functionality for designing Android applications. In the process of qualifying the bachelor's work were used: Android studio software, Kotlin documentation, XML international and national standards for building software systems, standards for building Android applications and user interfaces.

As a result of the bachelor's thesis, the goal was achieved and the following results were obtained:

An application has been created that implements the following functions:

• add contacts who use the application to the contact list;

• private chats;

• group chats;

• the ability to send files, pictures, voice messages.

The software implementation is made in the Kotlin programming language.

Keywords: Android application, messenger, firebase, Kotlin.

ЗМІСТ

ВСТУП

Актуальність дослідження. Мобільні телефони давно перестали бути просто засобом комунікації між людьми. Смартфони стали мінікомп'ютерами, які ми постійно носимо з собою. Найбільше смартфонів і планшетів випускаються на базі ОС Android. Найголовніші причини поширення саме цієї ОС: по-перше, Android це відкрита операційна система, по-друге, Android характеризується високою доступністю засобів розробки. Засоби розробки для платформи Android безкоштовні. Зазначені вище переваги обумовлюють масовість і широке поширення сучасних пристроїв на базі Android, оснащених різними функціями і додатками.

Оскільки в наш час мобільний телефон з ОС Android є майже в кожного. Починаючи з березня 2017 року ОС Android стала найпопулярнішою ОС, з якої виходили в інтернет. Так, 37,93% користувачів заходили в інтернет з Android, а з Windows — 37,91% користувачів. В Азії показники ще вищі — 52,2% і 29,2% відповідно. За останні 5 років відсоток пристроїв з ОС Android збільшилась, ця статистика буде розглядатися в наступних пунктах. З огляду на ці дані було прийнято рішення створити Android застосунок.

Отже, актуальність теми кваліфікаційної роботи бакалавра зумовлена важливістю дослідження та розроблення рішень, що допоможуть обмінюватися текстовими повідомленнями, файлами та голосовими повідомленнями.

Мета дослідження. Підвищення зручності спілкування між людьми з використанням розробленого мобільного застосунку.

Завдання дослідження. Відповідно до визначеної мети було поставлено наступні завдання:

1. Дослідити теоретичні основи побудови Android застосунків.
2. Створити мобільний застосунок з можливістю надсилати файли, голосові та текстові повідомлення в чати.
3. Розробити функціонал групових чатів.
4. Розробити користувацький інтерфейс для листування.

Об’єктом дослідження є листування в групах і приватних чатах з використанням Android застосунку.

Предметом дослідження є програмні підходи до побудови рішень для заповнення інформації, що зберігається в базі даних, особливості та основні принципи розроблення користувацьких інтерфейсів.

Методи дослідження: системний аналіз і синтез, вибір та застосування методу розробки мобільних застосунків, порівняння функціоналу IDE для конструювання Android застосунку.

У процесі виконання кваліфікаційної роботи бакалавра було використано: програмне забезпечення Android studio, документація Kotlin, XML міжнародні та національні стандарти з побудови програмних систем, стандарти з побудов Android застосунків та користувацьких інтерфейсів.

Практичне значення одержаних результатів полягає в створенні застосунку, за допомогою якого можна листуватися в приватних чатах і групах, а також обмінюватися файлами, картинками та голосовими повідомленнями.

Структура роботи. Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, поділених на підрозділи, висновків, списку використаних джерел та додатків. Загальний обсяг роботи складає 156 сторінок. Список використаних джерел охоплює 15 найменувань.

РОЗДІЛ 1. СУЧАСНІ ПІДХОДИ ДО РОЗРОБЛЕННЯ І ВПРОВАДЖЕННЯ ANDROID ЗАСТОСУНКІВ

1.1 Основні відомості про платформу Android

XXI століття – це століття інформаційних технологій. IT міцно увійшло в наше життя: вони оточують нас всюди. Інженери Apple, а потім і Google, знайшли рішення. Вони створили операційну систему для телефонів, що дозволяє з легкістю розробляти різноманітні застосунки, електронні бібліотеки, керування побутовими приладами, розумний дім.

Android – портативна операційна система для мобільних телефонів планшетних комп'ютерів, електронних книжок, наручних годинників, і багато інших типів пристроїв, що заснована на ядрі Linux.

Вихідний код Android знаходиться у відкритому доступі, тому будь-яка людина може завантажити його, змінити і опублікувати свою версію абсолютно безкоштовно. Цим Android відрізняється від закритих операційних систем, доступ до яких суворо обмежений, а розробники забороняють їх модифікувати.

ОС Android спочатку розроблялася компанією Android Inc., яку потім купила компанія Google. З травня 2017 року вона має більше 2 мільярдів активних користувачів. Google постійно покращує свої продукти, і Android – не виняток. З 2008 року компанія випустила 24 версії цієї ОС. Тепер кожен розробник електронного пристрою має можливість переробити Android під свій пристрій, таким чином гарантуючи сумісність свого устаткування зі сторонніми застосунками для цієї ОС. Більшість великих виробників пристроїв Android змінюють або доповнюють функції ОС на своїх телефонах і планшетах наприклад MIUI, Realme UI, Color OS, HarmonyOS, EMUI. Це виявилось дуже вигідним для розповсюдження Android. Установка його на телефони дала можливість з легкістю розробляти нові моделі мобільних пристроїв, розширюючи функціонал як телефонів, так і самої операційної системи. Поява програм, призначених для допомоги користувачеві в різних ситуаціях призвела до того, що на сьогоднішній день людині, коли вона вирушає в подорож, досить

просто мати з собою мобільний пристрій на ОС Android. Все це, разом з легкістю розробки застосунків, робить розглянуту платформу однією з найбільш перспективних для комунікації в сучасному суспільстві.

1.2 Актуальність розробки мобільних застосунків

Смартфони та інші мобільні пристрої не тільки стали частиною нашого повсякденного життя, вони – повноцінне продовження нас. Згідно зі статистикою, опублікованою statcounter.com, в квітні 2022 року 41,63% всіх пристроїв, з яких виходили в інтернет, базуються на ОС Android.

Актуальність розробки мобільних застосунків постійно зростає, це можна зрозуміти подивившись на розподіл ринку ОС (рис. 1.1). З кожним роком кількість пристроїв з ОС Android збільшується і поступово стає найпопулярнішою. Постійно з'являються застосунки, які ставлять нові рекорди популярності, наприклад нещодавна популярність TikTok.

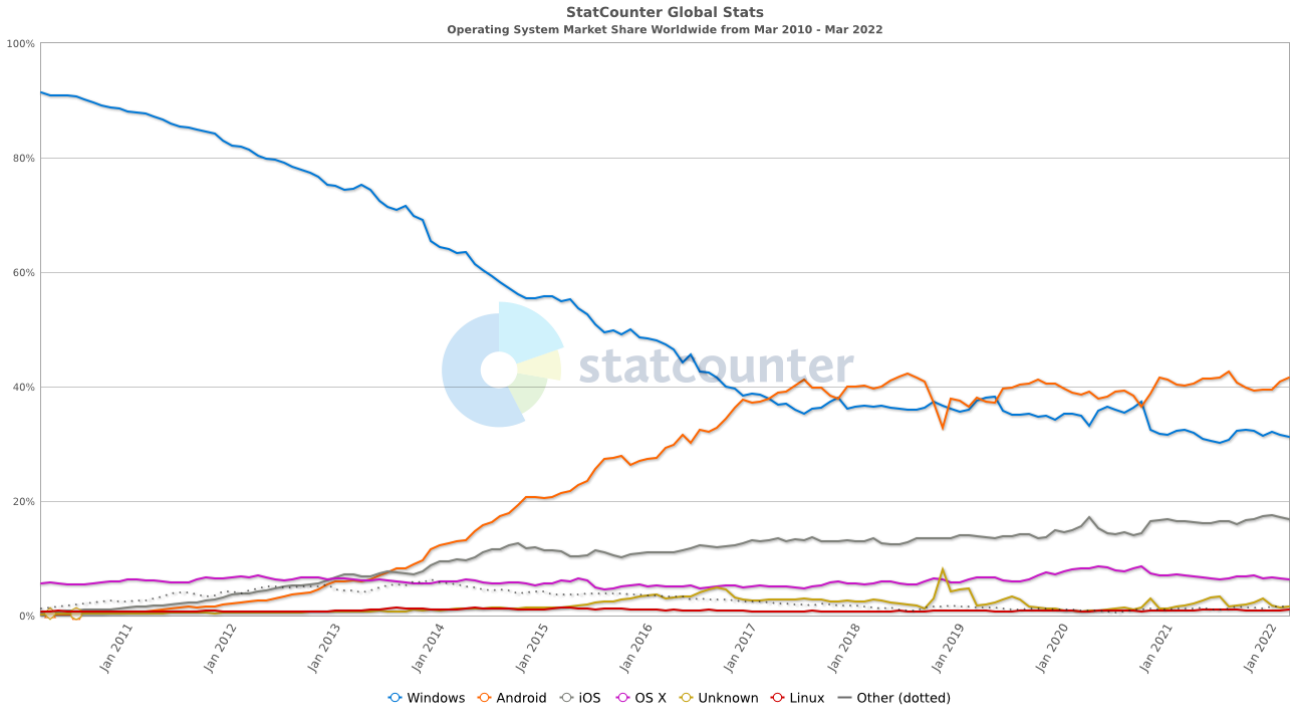

Рисунок 1.1 – Розподіл ринку операційних систем у світі

На рис. 1.2 можна побачити, що пристроїв на ОС Windows в Україні все ще більше, ніж на Android, але останні декілька років відрив зменшується і, дивлячись на загальносвітову тенденцію, можна спрогнозувати ще більше збільшення пристроїв на ОС Android.

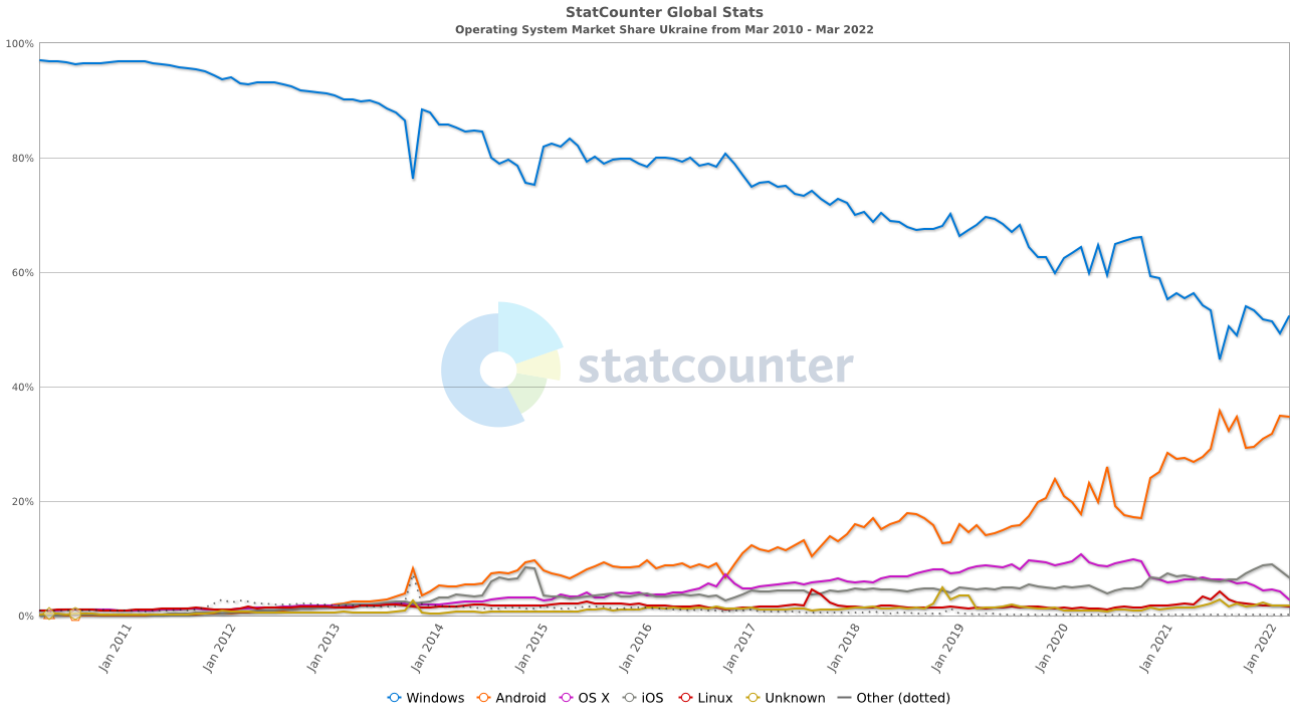

Рисунок 1.2 – Розподіл ринку операційних систем в Україні

1.3 Конкуренти ОС Android

Існує багато інших мобільних ОС, але на даний момент тільки iOS може складати конкуренцію, оскільки пристроїв на інших ОС набагато менше, ніж на Android. Причому ця тенденція спостерігається і в усьому світі, і в Україні, зокрема (рис. 1.3, 1.4).

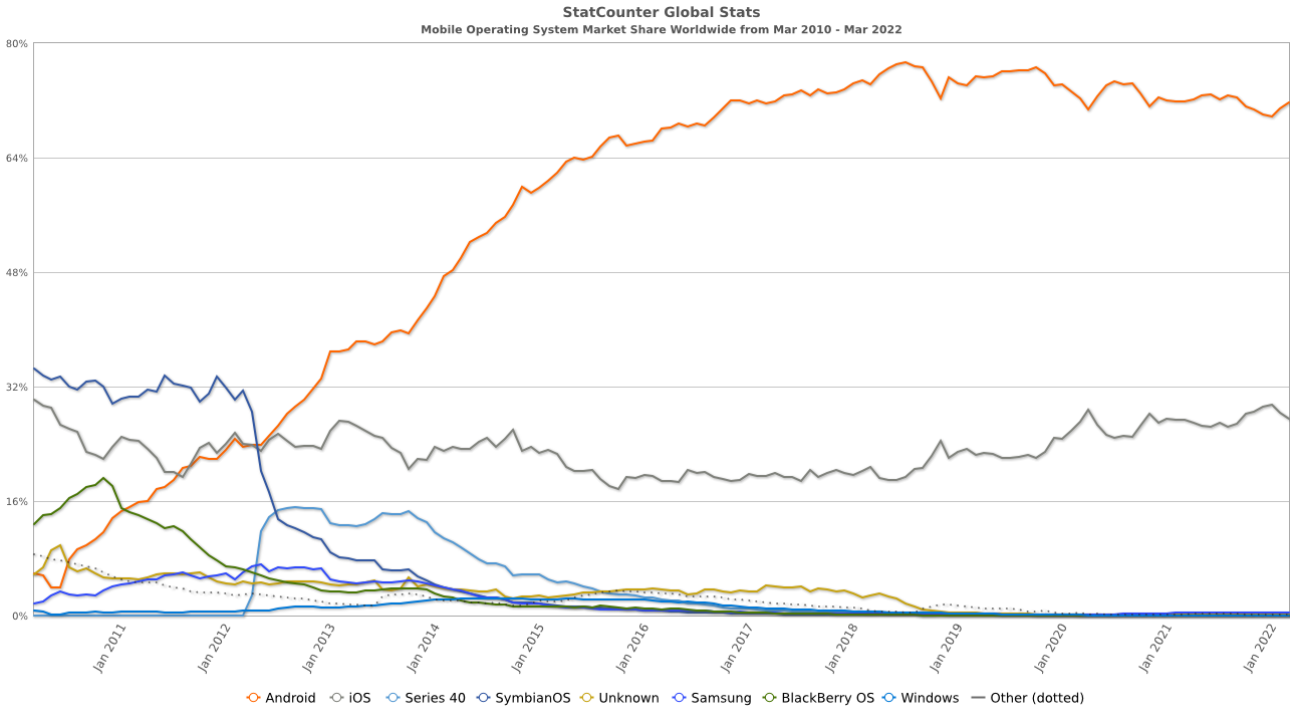

Рисунок 1.3 – Розподіл ринку мобільних операційних систем в світі

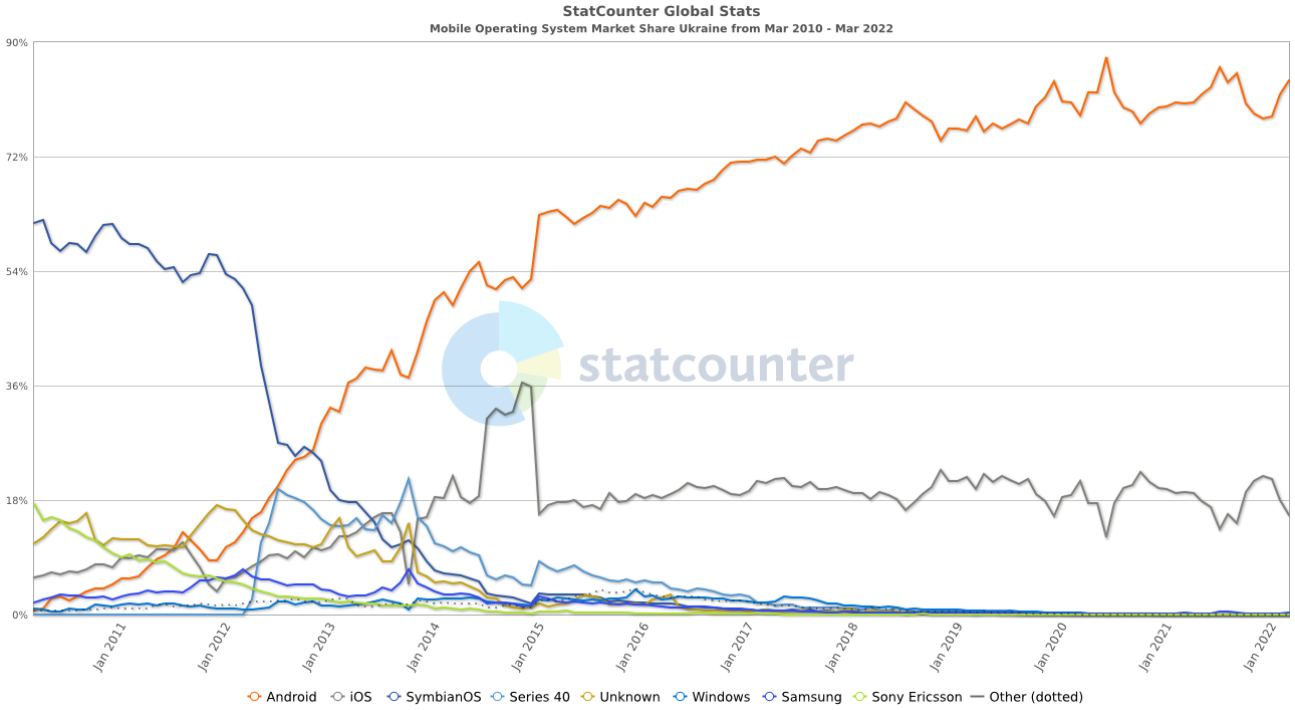

Рисунок 1.4 – Розподіл ринку мобільних операційних систем в Україні

Порівнюючи світову та українську статистику мобільних ОС (табл. 1.1) можна побачити, що Android, Windows і Series 40 – більш розповсюджені, а iOS, Samsung і KaiOS – менш розповсюджені в Україні.

Таблиця 1.1 – Порівняння розподілу ринку мобільних ОС в Україні та світі

	Android	iOS	Symbian OS	Series 40	Samsung	Windows	Інші
Україна	84.01%	15.64%	0.01%	0.02%	0.24	0.04%	0,04%
Світ	71,7%	27.57%	0,0%	0.01%	0.42%	0,01%	0,28%

Охарактеризуємо найпопулярніші ОС:

1. Apple iPhoneOS. Основний конкурент Android. iOS має єдиний інтерфейс для всіх застосунків і на відміну від Android, плавну анімацію, ексклюзивні програми та регулярні оновлення.

Серед плюсів варто відмітити зручний інтерфейс, плавну роботу, регулярні оновлення, відмінну якість застосунків та тривалу підтримку старих версій, а серед недоліків – це закрита ОС, висока вартість пристроїв.

2. KaiOS — це мобільний дистрибутив Linux для телефонів із клавіатурою. KaiOS в основному є веб-операційною системою, яка використовує застосунки, розроблені на веб-платформі HTML 5.

Серед плюсів варто відмітити “легкість” ОС, підтримку веб-застосунків і постійні оновлення, а серед недоліків не готовність до основного ринку смартфонів на Android.

3. Microsoft Windows Phone. Колись один з найбільших гравців на ринку мобільних ОС. Дана ОС не розроблюється, останнє оновлення 2 червня 2015 року. Причини цього в цілому, це бажання повторити модель Apple, але без втягування Microsoft в процес виробництва заліза: жорсткі вимоги до начинки апаратів, неможливість вносити зміни в програмне забезпечення, установка сторонніх застосунків тільки з фірмового магазину, замикання платформи на власні технології.

Плюси даної ОС – “легкість” ОС, а основні недоліки – припинена розробка даної ОС, закритий код ОС.

4. Tizen OS. мобільна операційна система на базі Linux , що підтримується Linux Foundation, в основному розроблена та використовується переважно Samsung Electronics. Схожа на android і має підтримку інтуїтивними жестами.

До плюсів даної ОС відноситься відмінна підтримка HTML5, підтримка інтуїтивних жестів, а недоліки – незначна підтримка сторонніх застосунків.

5. Ubuntu Touch - мобільна версія операційної системи Ubuntu , яка розробляється спільнотою UBports, оптимізована для смартфонів і планшетів.

Головні переваги даної ОС: відкритий код ОС, можливість функціонування, як робочий стіл, всі важливі застосунки встановлені за замовчуванням, але є і недоліки, а саме: розробка ведеться спільнотою, відсутність підтримки сторонніх застосунків на Android.

6. Harmony OS - розподілена операційна система, розроблена Huawei для співпраці та взаємозв’язку з кількома розумними пристроями.. Harmony OS підтримує Android застосунки. Ця ОС є копією ОС Android. Harmony ОС була розроблена через санкції на компанію Huawei.

1.4 Версії ОС Android

Google постійно покращує свої продукти, і Android – не виняток. З 2008 року компанія випустила 25 версій цієї ОС. Google анонсує серйозні поступові оновлення Android щороку. Остання основна версія – Android 12.

На рис. 1.5 можна побачити, що у світі більше 80% активних (згідно з статистикою statcounter.com) пристроїв використовують версію Android 8.0 і вище, а версії до 6.0 використовують менше 6,8% активних пристроїв.

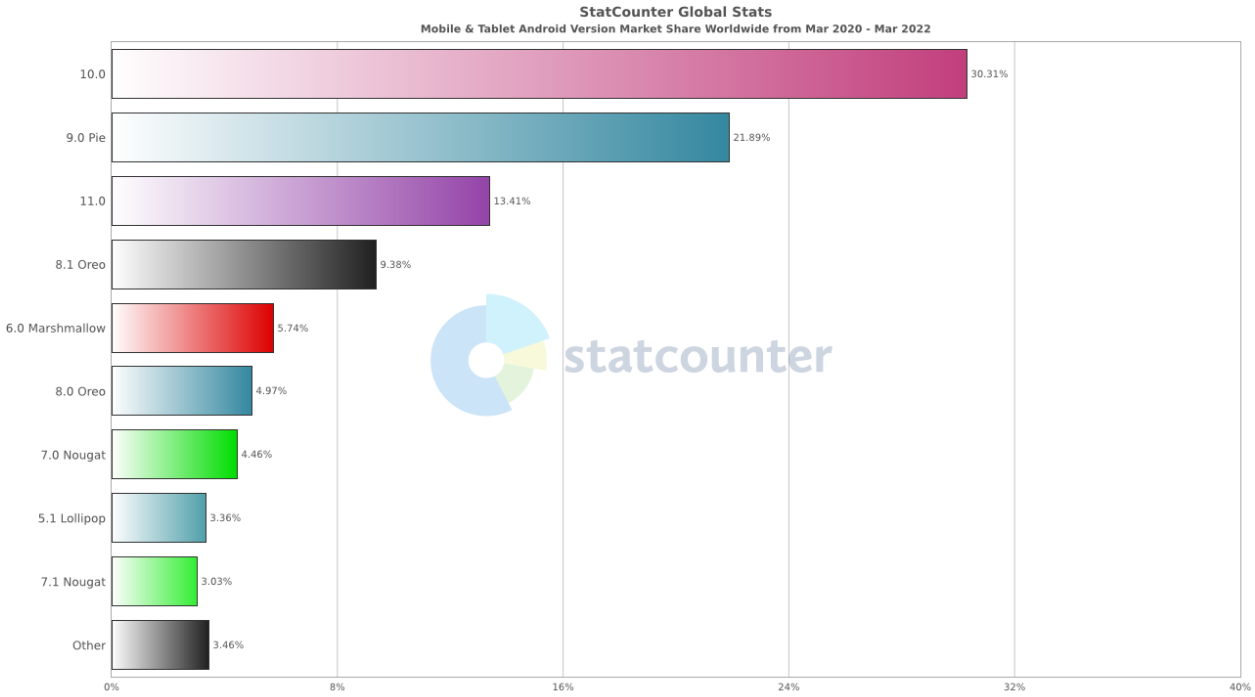

Рисунок 1.5 – Розподіл версій ОС Android в світі

На рис. 1.6 можна побачити, що в Україні більше 77% активних пристроїв використовують версію Android 8.0 і вище, а версії до 6.0 використовують менше 7,4% активних пристроїв. За цими даними можна зробити висновок, що українська статистика майже не відрізняється від загальносвітової тенденції.

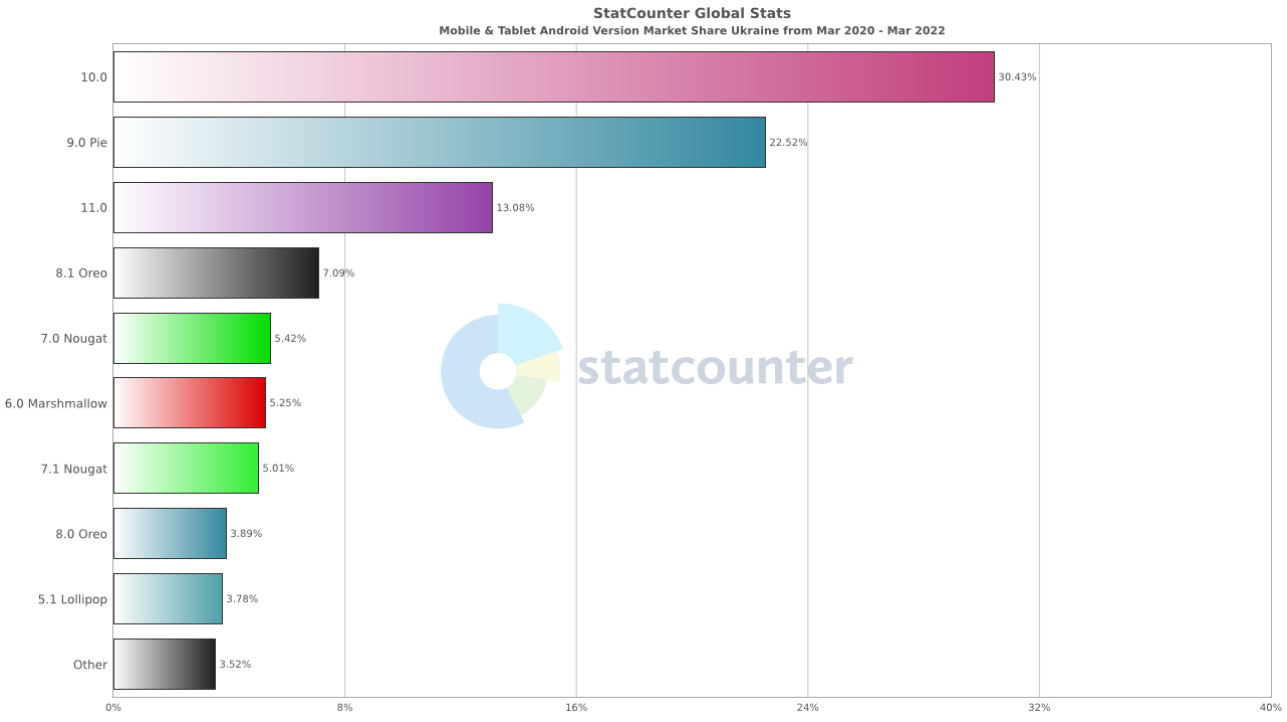

Рисунок 1.6 – Розподіл версій ОС Android в Україні

1.5 Google Play

Це платформа Google, яка пропонує своїм споживачам різноманітний цифровий контент. У 2012 році Google розпочав роз'єднання певних аспектів операційної системи, щоб їх можна було оновлювати через магазин Google Play незалежно від ОС.

Google Play — це офіційний магазин медіафайлів для Android, включаючи програми, ігри та електронні книги. Ви можете завантажувати вміст безпосередньо на пристрій Android через додаток Play Store або надсилати вміст на пристрій із веб-сайту Google Play. Обліковий запис розробника, який дає можливість публікувати застосунки, коштує 25$.

У Google Play можна знайти багато корисних і різноманітних застосунків. Google Play став одним з чинників популярності Android, оскільки він став зручним інструментом розповсюдження застосунків для розробників і зручним для пошуку різноманітних застосунків для користувачів.

1.6 Kotlin

Kotlin — це мова програмування загального призначення, розроблена для повної взаємодії з Java. Творці Kotlin спробували взяти найкращі можливості з JavaScript, TypeScript, Java, C#Rust та багатьох інших мов, щоб створити мову, яка компілювалася так само швидко, як Java, але була б кращою в усіх інших функціях.

У 2019 році Google оголосив, що Kotlin став його улюбленою мовою для розробників застосунків Android. Хоча, порівняно з Java, Kotlin все ще набагато менш популярний, займаючи 28-е місце в індексі TIOBE і 12-е, згідно з індексом PYPL.

Переваги Kotlin:

- Байт-код — це скомпільований формат, який використовується для програм Kotlin. Це означає, що коли розробник програмує код і збирає його, він запускається через віртуальну машину, а не через процесор комп'ютера.
- Найкраща перевага використання мови Kotlin полягає в тому, що ви все ще можете використовувати всі існуючі бібліотеки та фреймворки Java. Крім того, вдосконалені фреймворки також покладаються на обробку анотацій. Плюсом Kotlin є те, що розробник може легко інтегрувати Kotlin з Maven, Gradle та іншими системами збірки.
- За допомогою нової функції, інтегрованої JetBrains в IntelliJ, розробник може легко конвертувати Java в Kotlin і заощадити час. Крім того, вам не потрібно повторно вводити звичайний код для Kotlin.
- З Kotlin вам не доведеться турбуватися про NullPointerExceptions. Система Kotlin допоможе нам уникнути винятків з нульовим покажчиком. Якщо порівняти Java з Kotlin, система Kotlin легко відмовляється компілювати код, який призначає або повертає null.

Обмеження Kotlin:

- Kotlin має меншу продуктивність компіляції. Це головне обмеження Kotlin. За деяких обставин Kotlin перевершує Java за швидкістю компіляції, але часто вона значно нижча.
- Програмістам, які володіють мовою Java, також важко перейти на Kotlin. Тому для організації навчання Kotlin можуть знадобитися додаткові витрати для компанії.
- Виконуючи інкрементні збірки, Kotlin працює швидко. Однак, щоб розробити чисту збірку для програм Android, Java є переможцем.
- Kotlin існує вже 6 років. Більше 8% розробників використовують Kotlin. Через обмежені ресурси та підтримку з боку спільноти розробникам важко вирішити проблеми в процесі розробки.

Переваги Java:

- Java є об'єктно-орієнтованою мовою програмування і легше створювати модульні програми та код.

- Java досить гнучка для переходу з однієї комп’ютерної системи на іншу. Це незалежна від платформи мова.
- Java — це незалежна від платформи мова; вона включає функцію WORA (Write Once Run Anywhere).
- Java — це багатопотокова мова. Це означає, що програма дозволяє одночасно виконувати декілька частин програми і максимально використовує ЦП.

Обмеження Java:

- Java може споживати пам'ять і призвести до низької продуктивності навіть у порівнянні з C++ або C. Причина в тому, що Java потрібно конвертувати в код машинного рівня.
- Java зосереджується лише на сховищі, а не на резервному копіюванні даних.
- Конструктори графічного інтерфейсу Java все ще не є надійними та зрілими. Для розробки UI використовуються відомі фреймворки, такі як SWT, JavaFX, JSF. Можливо, вам доведеться витратити додатковий час на дослідження, перш ніж вибрати правильний конструктор графічного інтерфейсу.
- Unsigned int, unsigned char не підтримується в Java. Однак API без знака int і unsigned long можна знайти в Java 8.

РОЗДІЛ 2. АНАЛІЗ АРХІТЕКТУРНИХ РІШЕНЬ І ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ

2.1 Розроблення мобільних застосунків

Розроблення мобільних застосунків для смартфонів, планшетів або будь-яких інших мобільних пристроїв – це написання програмного коду з метою створення програм, які будуть працювати на певних платформах. Одне із найголовніших – це інтерфейс застосунку, оскільки інтерфейс з'єднує апаратну частину з програмним забезпеченням мобільного пристрою і повинен бути зручним і зрозумілим для користувача. Під час розроблення застосунку потрібно враховувати розміри елементів інтерфейсу, оптимізувати розміщення елементів, зробити їх інтуїтивно зрозумілими і компактно розмістити. Тому завдання – це не тільки написати програмний код.

Важлива особливість розроблення мобільних застосунків – це вигляд застосунку на пристроях з різним форм-фактором, тому що це можуть бути як смартфони, планшети, які мають великі екрани з різними відношеннями сторін, так і розумні годинники, які мають маленький екран. Тому дуже важливим моментом створення мобільного застосунку є забезпечення адаптивного інтерфейсу.

Також під час розробки потрібно враховувати тип пристроїв, для яких буде розроблятись застосунок. Оскільки якщо застосунок буде розрахований не тільки для смартфонів і планшетів, а також і для інших пристроїв, з цим краще визначитися заздалегідь. Щоб застосунок коректно працював на різних мобільних пристроях потрібно передбачити різні способи взаємодії з цим застосунком і різні способи відображення інформації.

2.2 Інструментарій Android

Спочатку для розроблення застосунку потрібно обрати мову програмування. Для ОС Android в більшості випадків це Java, проте останнім часом набирають популярність Kotlin і прогресивний новачок Dart.

Для написання застосунку, передбаченого завданнями даної дипломної роботи було обрано мову Kotlin.

Для створення програм мовою Kotlin необхідне спеціальне програмне забезпечення такі інструменти як JRE (Java Runtime Environment) і JDK (Java Development Kit). JRE це середовище виконання в якому виконується код. JDK – це набір інструментів. Також для розробки потрібно вибрати IDE, для розроблення застосунку було обрано Android Studio. Для завантаження Android Studio потрібно перейти на офіційний сайт розробника.

2.3 Етапи та методи розроблення мобільних застосунків

Для початку потрібно обрати підхід до розроблення, наприклад, HCD (Human centred design), що означає орієнтацію на проблемах користувача та покращення інтерфейсу.

Можна виділити такі стадії розробки Android застосунку:

1) Пошук ідеї для застосунку. Для початку роботи над застосунком потрібно визначитись з ідеєю, проаналізувати ринок на наявність схожих застосунків. Визначити, що в них подобається і чого не вистачає, що потрібно додати, а що відкинути, що можна вдосконалити.

2) Визначення базового функціоналу. Після виконання попереднього пункту у вас повинно бути розуміння мінімального функціоналу (скелет проекту), на який можна буде додавати новий функціонал. Також потрібно визначити набір базових можливостей доступних користувачеві.

3) UI / UX design. На цьому етапі повинно бути розуміння функціоналу застосунку і виходячи з нього можна уявити як користувач буде використовувати застосунок, потрібно створити структуру навігаційної системи програми. Створюється прототип взаємодії користувача з програмою, тобто визначається

як саме буде працювати застосунок. Це не обов’язково повинен бути кінцевий дизайн, це можуть бути просто макети сторінок застосунку.

4) Перевірка прототипів в тестах. Перевірка нашого зачатку застосунка на практиці, тобто перевірка чи була досягнена початкова ціль застосунку, перевірка зручності використання.

5) Розробка мобільного застосунку. Якщо вас все влаштовує на попередньому кроці, можна починати реалізовувати прототип в програмному коді.

6) Тестування готової версії застосунку. На цьому етапі потрібно проводити тестування застосунку на пошук багів, оцінити зручність готового дизайну.

7) Виправлення недопрацювань. Виправлення багів і недопрацювань, знайдених при тестуванні.

8) Реліз. Після виправлення всіх недопрацювань наш застосунок готовий до релізу. Його можна завантажити до магазину додатків (Google Play).

Існує декілька основних методів розробки мобільних застосунків:

- нативна (Native);
- кросплатформна (Cross-Platform);
- гібридна (Hybrid).

Нативна розробка – це розробка застосунку для однієї конкретної ОС, використовуючи власні стеки технологій кожної операційної системи, щоб досягти найкращої продуктивності. Плюси такої розробки – швидка реакція на дії користувача, доступ до апаратної частини конкретного пристрою.

Для розробки кросплатформних застосунків використовується один із багатьох кросплатформних інструментів розробки, що дозволяють компілювати одну і ту ж базу коду для різних операційних систем. В основному розробка кросплатормних застосунків здійснюється з використанням web-технологій (JavaScript, HTML, CSS), які дозволяють розробити застосунок одразу на кілька платформ. Але для коректрої роботи на певній платформі потрібно використовувати певні інтерпритатори, що в свою чергу вплине на швидкодію.

Це швидше, ніж нативна розробка, але оптимізація мобільних застосунків для декількох платформ може бути складною і не завжди ефективною.

Під час розроблення гібридного застосунку поєднуються нативні і веб-інструменти, щоб створити застосунок, який буде працювати на декількох мобільних платформах.

2.4 Firebase

Firebase — це платформа для розробки, яка стала відомою своєю базою даних у реальному часі, яка все ще є основою багатовузлової бази даних ключ-значення, оптимізованої для синхронізації даних, часто між машинами користувачів або смартфонами та централізованим сховищем у хмарі. Вона розроблена, щоб полегшити життя розробникам, обробляючи більшу частину передавання та витягування даних. Це звільняє розробників програм від тягаря програмування, пов’язаного з керуванням версіями або розташуванням. Вони можуть записати нові біти до Firebase, і дані будуть узгоджені в усій системі.

Firebase цінується значною мірою тому, що він може постійно поширювати та синхронізувати зміни між локальними копіями інформації, що зберігаються на машинах користувачів, із версіями, що зберігаються в хмарі. Firebase усуває багато проблем, пов’язаних із змішуванням аутентифікації, синхронізації та сегрегації, поєднуючи декілька версій і забезпечуючи однакові правильні біти у всій системі.

Ця платформа – набір певних сервісів, які включені у продукт Firebase. Там ви можете побачити безліч сервісів: Cloud Firestore, Cloud Functions і те, про що ми говоритимемо в цьому відео – це Realtime Database (база даних у режимі реального часу).

2.5 Додаткові бібліотеки

Navigation drawer

Навігаційні ящики надають доступ до місць призначення та функцій програми, наприклад перемикання облікових записів. Вони можуть постійно відображатися на екрані або керуватися за допомогою значка навігаційного меню.

Навігаційні ящики рекомендуються для:

- Програми з п’ятьма або більше адресами верхнього рівня
- Програми з двома або більше рівнями ієрархії навігації
- Швидка навігація між непов'язаними пунктами призначення

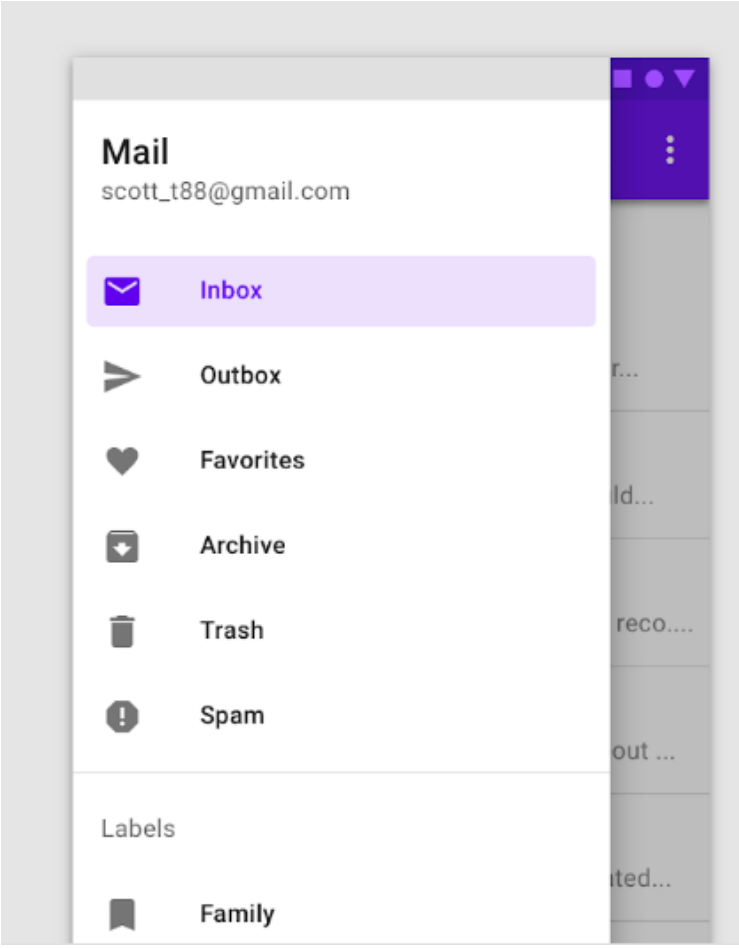

Рисунок 2.1 – Приклад використання Navigation drawer

CircleImageView

Швидкий круговий ImageView ідеально підходить для створення зображень профілю. Базується на RoundedImageView і використовує BitmapShader.

Можливості:

- створити копію оригінального растрового зображення
- використовувати clipPath (який не має ані апаратного прискорення, ані згладжування)
- використовувати setXfermode, щоб обрізати растрове зображення (що означає двічі намалювати на полотні)

Оскільки це лише користувальницький ImageView, а не користувацький Drawable чи комбінація обох, його можна використовувати з усіма видами малюнків, тобто PicassoDrawable від Picasso або іншими нестандартними файлами для малювання

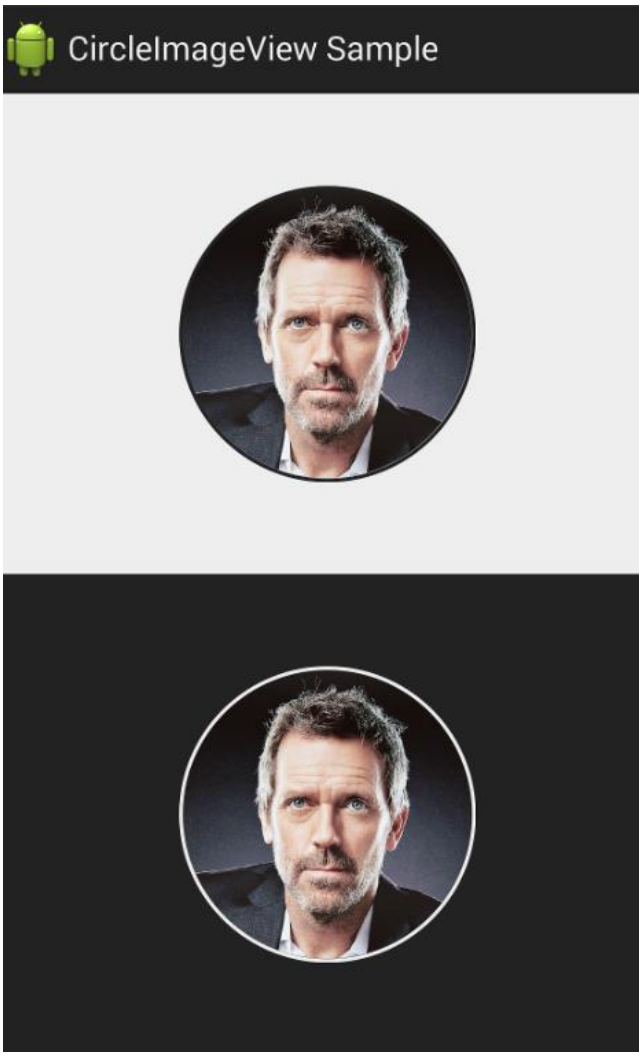

Рисунок 2.2 – Приклад використання CircleImageView

Android Image Cropper

- **Потужна** (масштабування, поворот, багатоджерела)
- **Настроювана** (форма, межі, стиль)
- **Оптимізована** (асинхронізація, вибірка, матриця)
- **Проста** бібліотека для обрізання зображень для Android.

Особливості

- Вбудований CropImageActivity.
- Встановіть обрізане зображення як Bitmap, Resource або Android URI (галерея, камера, Dropbox тощо).
- Поворот/перегортання зображення під час кадрування.

- Автоматичне збільшення/зменшення масштабу до відповідної області обрізання.
- Автоматичний поворот растрового зображення за даними Exif зображення.
- Встановити мінімальні/максимальні межі зображення результату в пікселях.
- Встановити початковий розмір/розташування вікна обрізання.
- Запит на зміну розміру обрізаного зображення до певного розміру.
- Оптимізація пам'яті растрового зображення, обробка OOM (ніколи не повинно відбуватися)!
- Рівень API 14.

Налаштування

- Форма вікна обрізання: прямокутна або овальна (куб/коло з фіксованим співвідношенням сторін).
- Співвідношення сторін вікна обрізання: безкоштовно, 1:1, 4:3, 16:9 або на замовлення.
- Зовнішній вигляд інструкцій: Вимкнено / Завжди ввімкнено / Показати на Toch.
- Вікно обрізання. Лінія кордону, кут межі, товщина та колір орієнтирів.

Корутини

Coroutine (корутини), або співпрограми - це блоки коду, які працюють асинхронно, тобто по черзі. У потрібний момент виконання такого блоку зупиняється із збереженням всіх його властивостей, щоб запустився інший код. Коли керування повертається до першого блоку, він продовжує роботу. В результаті програма виконує кілька функцій одночасно.

Для чого потрібні корутини

- Для створення асинхронних програм, які можуть виконувати кілька дій одночасно.
- Для гнучкої та зручної реалізації багатозадачності.
- Для більшого контролю під час перемикання між різними завданнями. Корутинами управляють розробник та програма, а не операційна система.
- Зменшення навантаження на апаратні ресурси пристрою.

Супрограми - це потоки виконання коду, які організуються поверх системних потоків. Потік виконання коду - це послідовність операцій, що виконуються один за одним. Вона виконується, доки настане момент, який заданий у коді чи визначено розробником. Потім може почати виконуватись частина іншої послідовності операцій. У системних потоках містяться інструкції процесора, одному його ядрі можуть по черзі працювати різні системні потоки. Корутини працюють на вищому рівні: кілька співпрограм можуть послідовно виконувати свій код на одному системному потоці.

Для кінцевого користувача робота корутин виглядає як кілька дій, які виконуються паралельно.

Принципи роботи корутин:

- Можуть мати кілька точок для входу та повернення.
- Припиняються та відновлюють роботу більше одного разу.
- Діють за стратегією LIFO — остання викликана корутина закінчиться першою.

Відмінності між корутинами та потоками

Багатозадачність з використанням корутин легко переплутати із багатопоточністю. Це виконання програми у кількох системних потоках.

Потік - складова частина процесу, що виконується в операційній системі. Принцип схожий: кілька потоків зупиняються та відновлюються, чекають один на одного, спілкуються. Але є відмінності.

- Потоками керує операційна система. Перемикання корутин - розробник за допомогою коду.
- Перемикання потоків складно контролювати. Корутини контролюються гнучкіше.
- Потоки забирають багато ресурсів процесора - йому постійно доводиться перемикатися між ними. Корутини не вимагають перемикання контексту, тому код споживає мало ресурсів.
- Потоки виконуються на апаратному чи системному рівні. Корутини – більш високорівневе рішення. Це означає, що вони далі від системи та апаратних ресурсів, проте ближче до людських понять.
- Потоки прискорюють виконання складного завдання, але забирають багато ресурсів. Корутини не підвищують швидкість, але допомагають оптимізувати навантаження.
- Корутини виконуються у межах одного потоку або пулу потоків.

Переваги використання корутин

- Ефективність
- Зручність для користувача
- Зниження навантаження на систему
- Гнучкість в управлінні

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ANDROID ЗАСТОСУНКУ

3.1 Архітектура і реалізація

Застосунок для Android складається з активностей, кожній з них відповідає вікно керування, кожна активність представлена класом реалізованим мовою Java або Kotlin, кожній активності відповідає свій xml файл який описує дизайн в якому за допогою xml-коду описане розташування об’єктів. За допомогою характеристик заданих в xml-файлі задається розмір об’єктів і виводяться на екран. Тому на будь-якій діагоналі активність буде виглядати однаково. Також в проекті існує manifest.xml в якому прописані залежності, версії модулів і мінімальні вимоги до системи.

На рис. 3.1 представлена діаграма роботи фрагмента реалізованого застосунку, що демонструє відправлення повідомлення в чат. У сфері розроблення програмного забезпечення використовується спеціальна мова для графічного опису об’єктного моделювання – мова UML (Unified Modeling Language). При опису роботи програми створюється абстрактна модель системи або підсистеми, яка називається UML-моделлю. На стадії опису роботи програми для наочного уявлення роботи окремих функцій програми наводиться Діаграма компонентів. Оскільки основними елементами програми операційної системи Android є активності, то схема роботи реалізованого застосунку являє собою схему зв’язків між активностями.

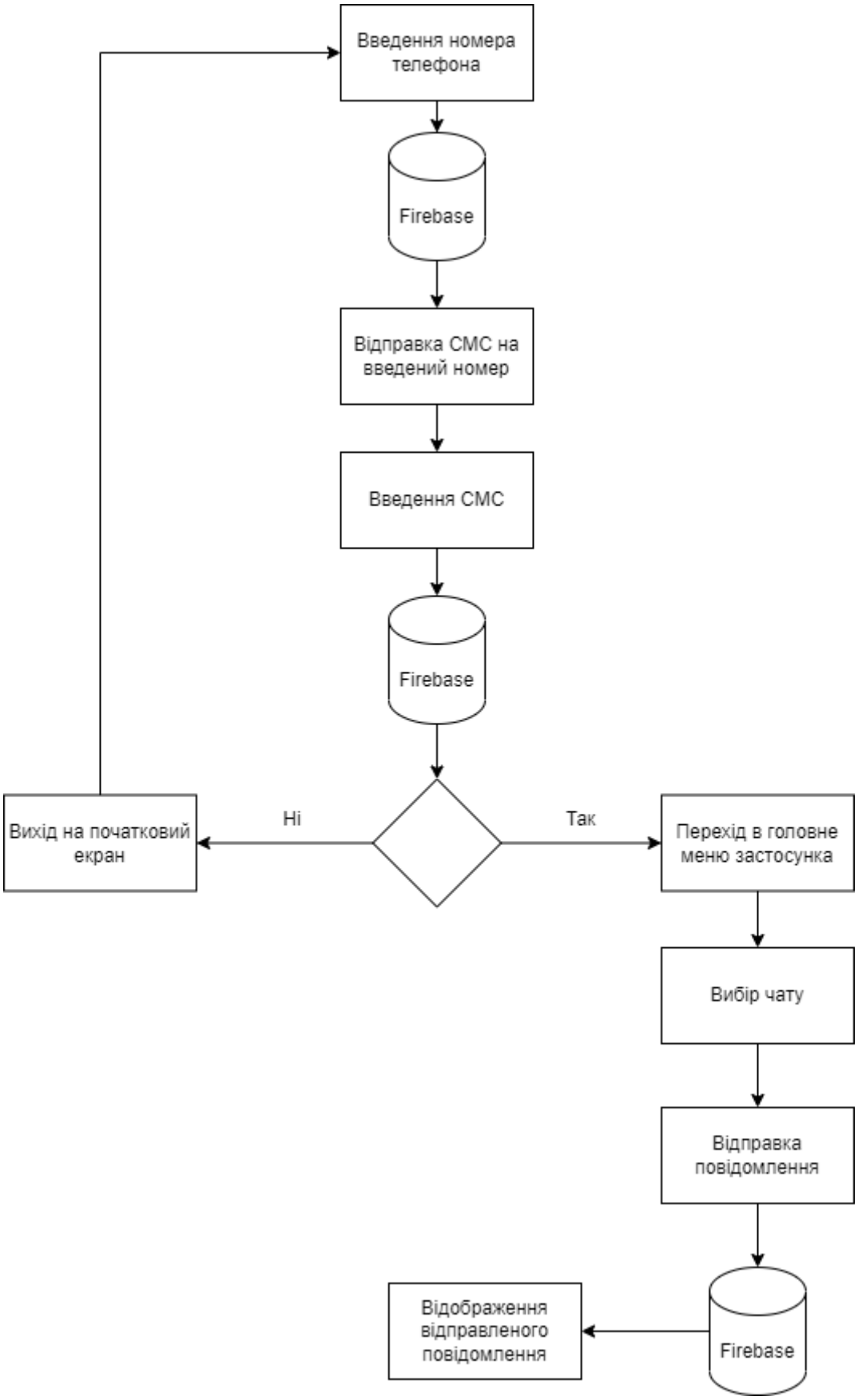

Рисунок 3.1 – Покрокова схема роботи фрагмента застосунку

На рис. 3.2 представлена діаграма потоків даних. Схеми поширення запитів вказані лініями, а потоки даних вказані стрілками.

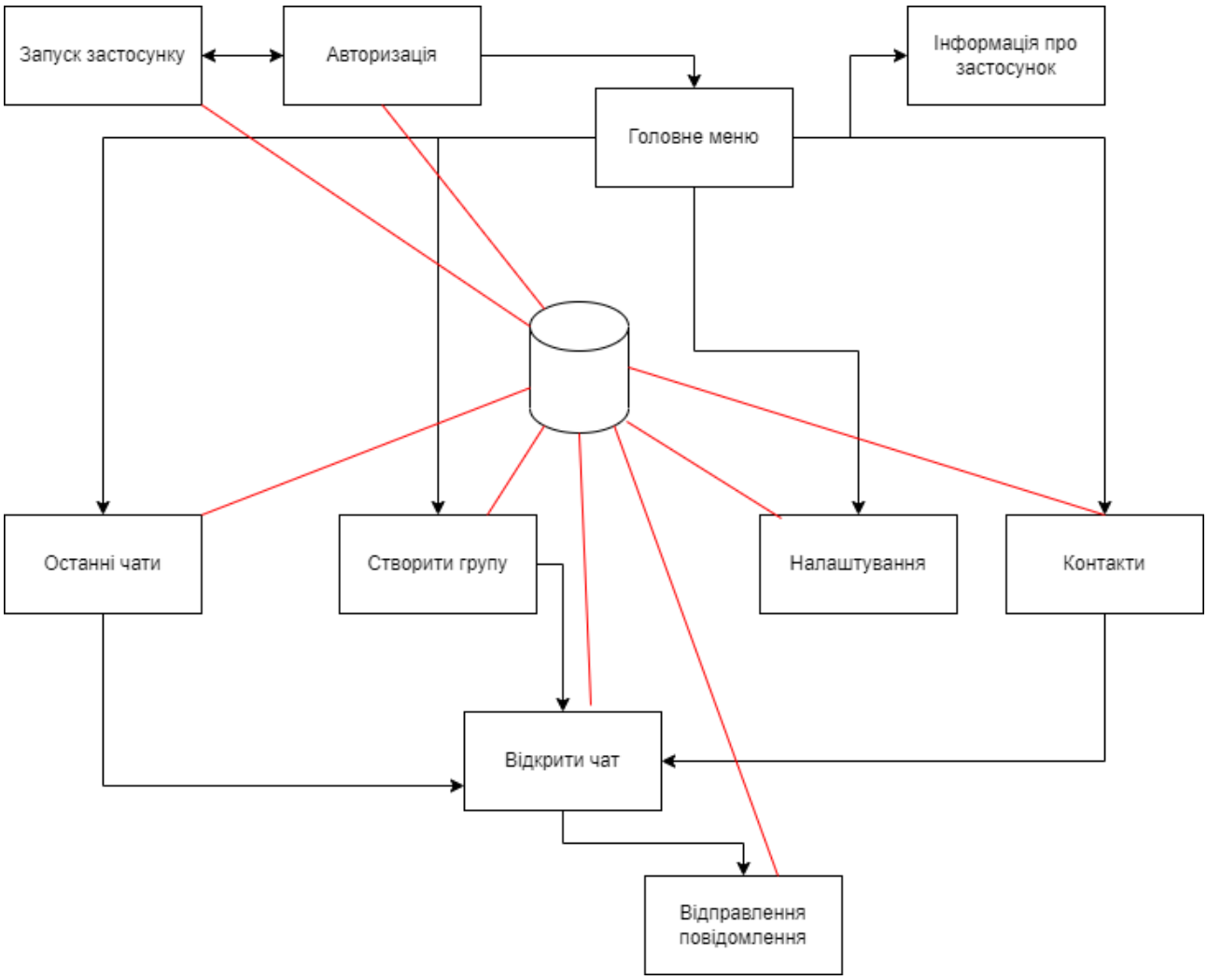

Рисунок 3.2 – Діаграма потоків даних застосунку

При завантаженні програми основна активність звертається до бази даних, Firebase. Після цього користувачу потрібно авторизуватися. Після входу відкривається головне меню, де користувач може обрати потрібну йому категорію. При виборі категорії в меню завантажується активність відповідна вибраній, наприклад чати, створення груп, налаштування, інформація про програму, яка звертається до бази даних і, прочитавши можливі категорії, відображає їх у списку. При виборі останніх чатів, контактів завантажується активність «Список об’єктів». За допомогою механізму передачі параметрів між активностями їй передають всі елементи цієї категорії і відображається невелика

форма, яка складається з назви чату або контакта, картинки та часу останнього повідомлення. Активність являє собою RecycleView, в одному з табів якого знаходиться картинка, а в інших – текстові поля. Активність запитує з бази даних інформацію про об’єкт і заповнює текстові поля.

Для проекта була вибрана база даних Firebase, так як вона є вбудованою в Android Strudio, вона безкоштовна і для неї є багато документації.

Для створення головного меню (Navigation Drawer) це видвижне меню з хідером з інформацією про аккаунт і нижньою частиною з списком категорій. Для зручної реалізації цього була вибрана бібліотека Material Drawer. Далі для реалізації дій при воборі категорії в меню потрібно створити фрагмент для кожного пункту в меню InfoFragment, ChatFragment…

Для відображення круглих фото в профілях і чатах використовується CircleImageView.

Для зручного обрізання фото при зміні фото профілю буде використовуватись бібліотека Android Image Cropper.

Для полегшення роботи з картинками використовується бібліотека Picasso.

Для оптимізації виконання деяких функцій використовуються Корутини, за допомогою них деякі завдання виконуються в фоновому режимі.

3.2 Список модулів

Функціонально застосунок складається з наведених нижче модулів (активностей). Активність є схемою представлення Android застосунків. Кожен екран для користувача інтерфейсу представлений класом Activity і по суті є окремою формою застосунку. Android застосунок може складатися з декількох активностей і може перемикатися між ними під час виконання програми.

• **Активність з полем для введення номера телефона (Fragment enter phone number)** Активність з полем для введення номера телефона і кнопкою для переходу на іншу активність з введенням коду (рис. 3.3). При натисканні на кнопку на введений номер відправляється смс з кодом.

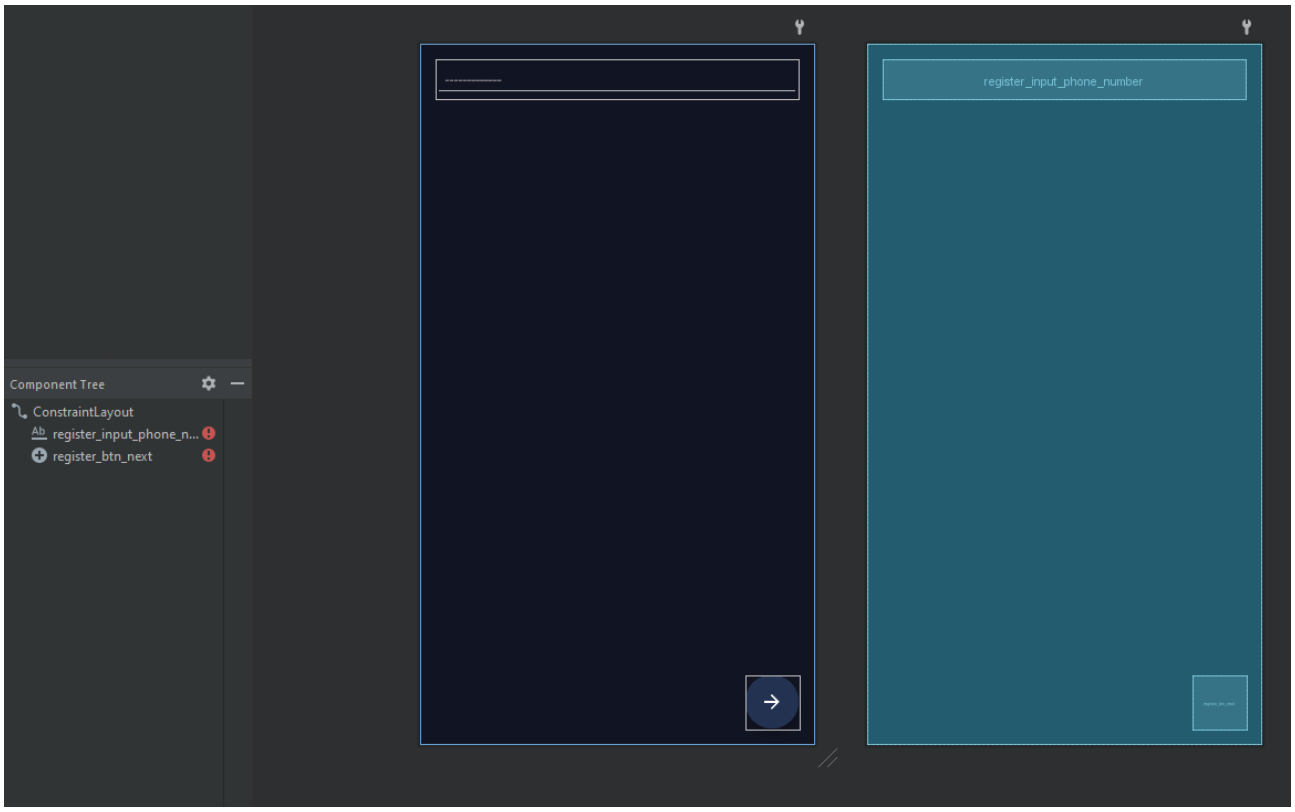

Рисунок 3.3 – Активність з полем для введення номера телефона

• **Активність з полем для введення коду (Fragment enter phone number)** Активність з полем для введення коду отриманого в смс (рис. 3.4). При введенні правильного коду відкривається наступна активність Main Activity.

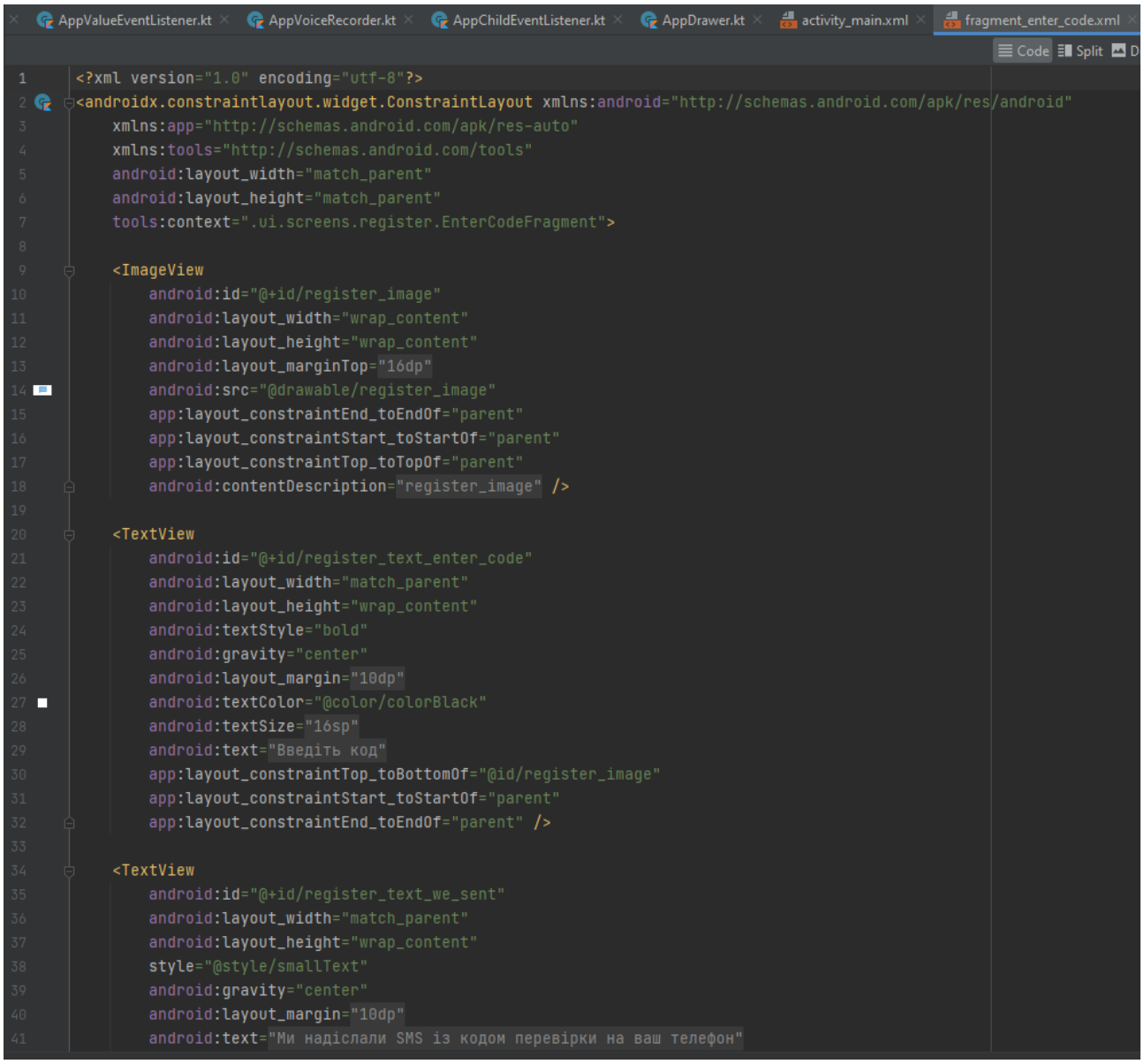

Рисунок 3.4 – Активність з полем для введення коду

• **Основна активність (Activity Main)** (рис. 3.5) містить головне меню з аватаркою, ім’ям користувача, номером телефона, меню зі списком об’єктів для вибору і списком останніх чатів (Fragment main list), який складається з об’єктів активності (Main list) (рис. 3.6-3.7). Вона призначена для зручної навігації або для вибору останніх чатів не витрачаючи час на відкриття інших об’єктів. Активність використовує шаблон форматування ConstraintLayout.

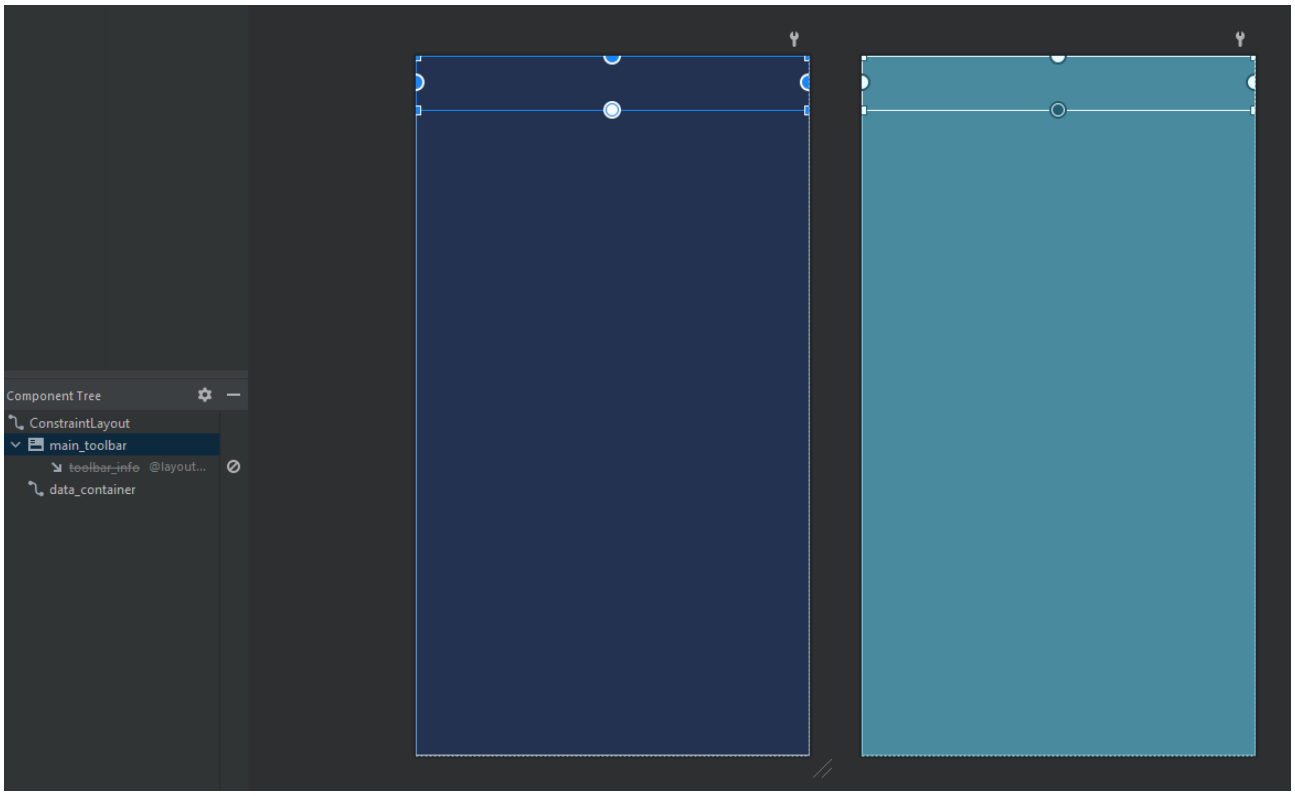

Рисунок 3.5 – Основна активність

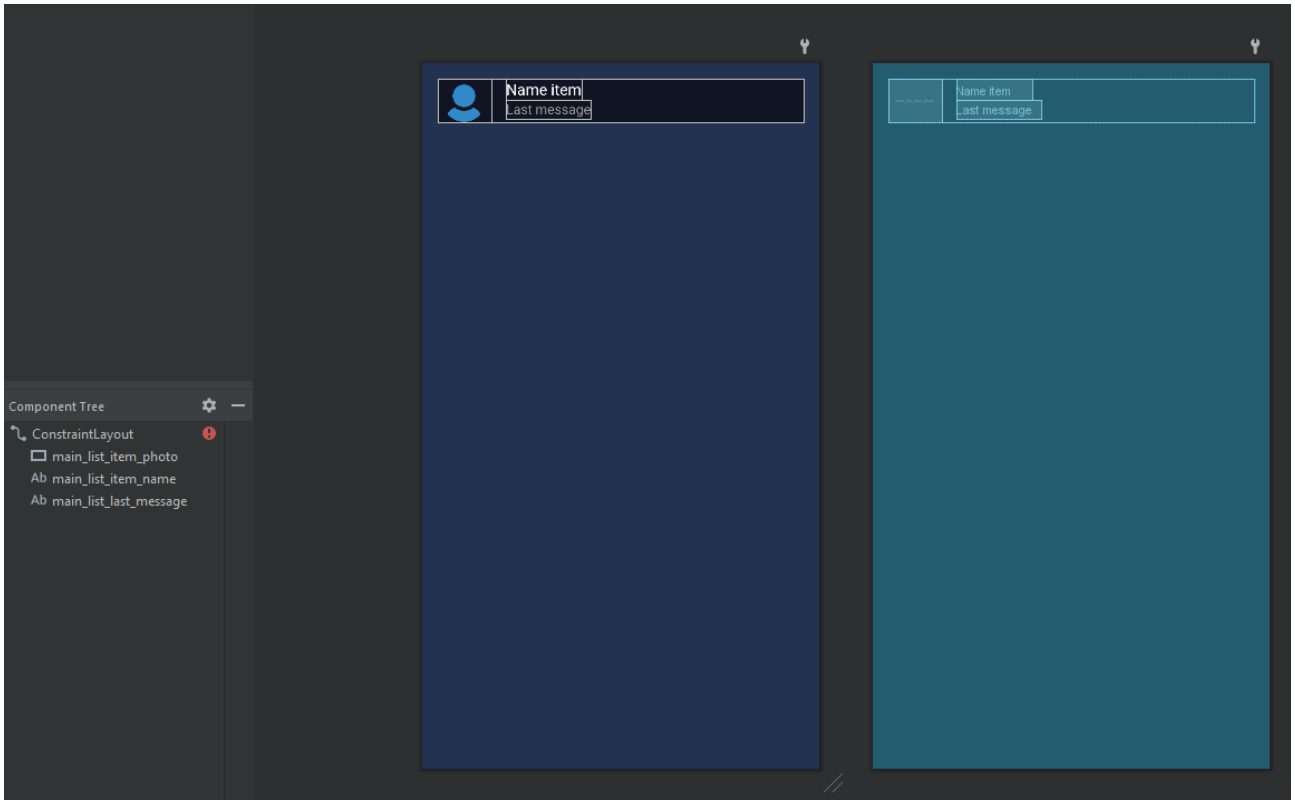

Рисунок 3.6 – Головний список

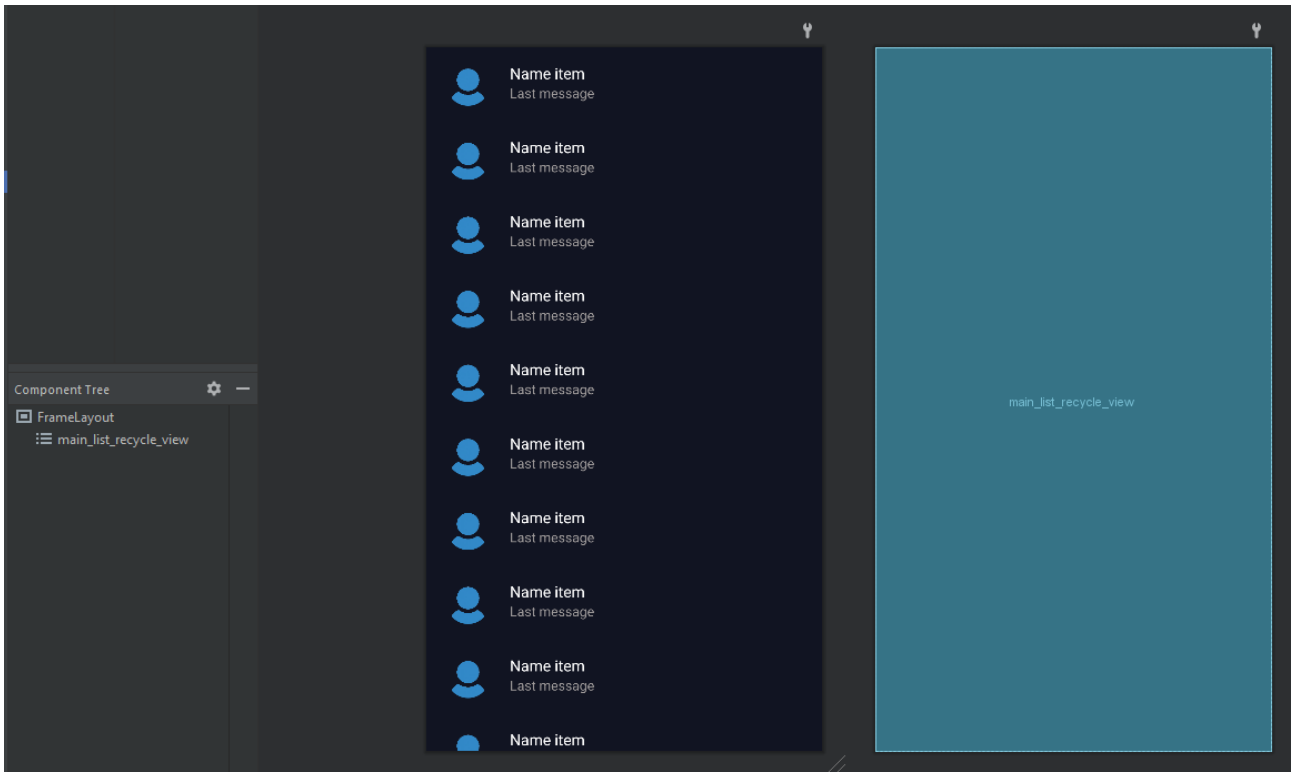

Рисунок 3.7 – Список об'єктів головоної активності

• **Об’єкт в списку контактів (Contact item)** (рис. 3.8). Активність використовує шаблон форматування ConstraintLayout. Вона відповідає за відображення контактів в списку контактів.

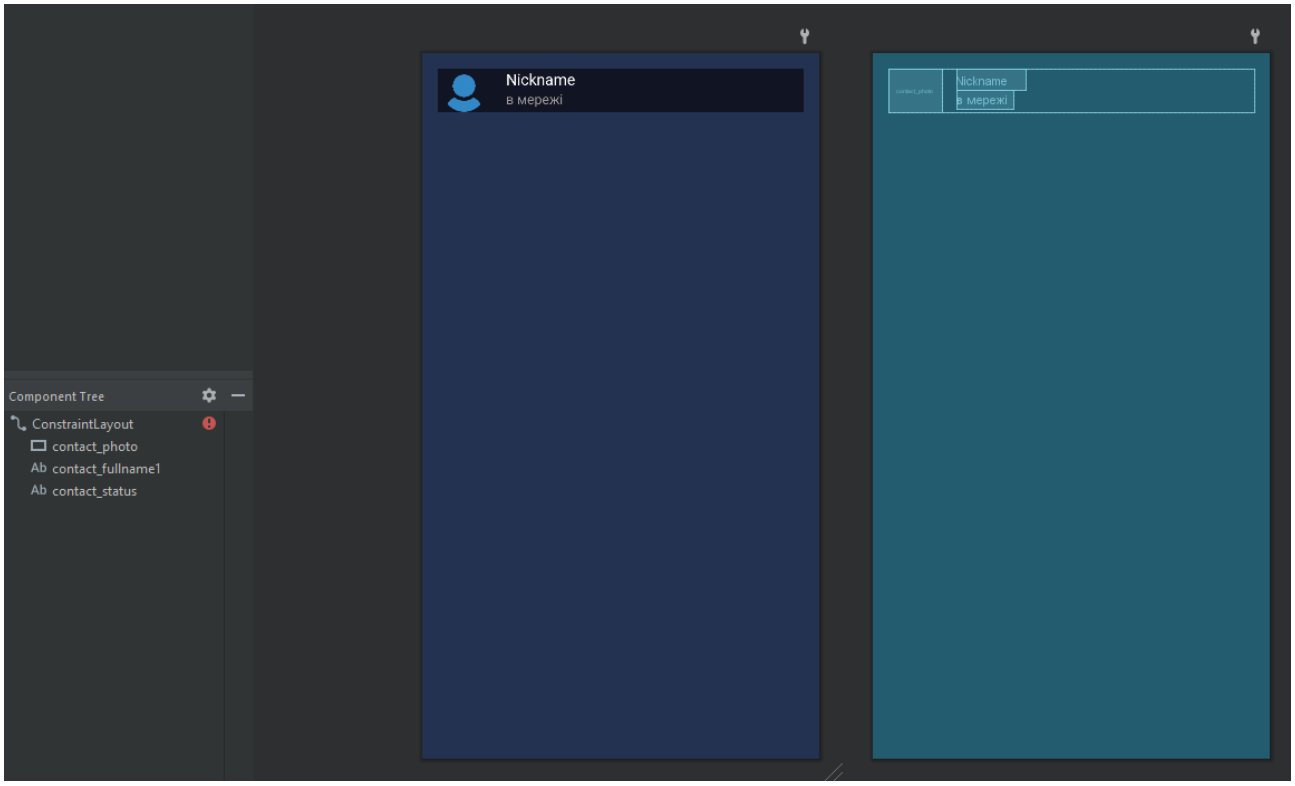

Рисунок 3.8 – Об’єкт в списку контактів

• **Об’єкт для додавання контакта (Add contact item)** (рис. 3.9). Активність використовує шаблон форматування ConstraintLayout. Вона відповідає за відображення контакта при його додаванні.

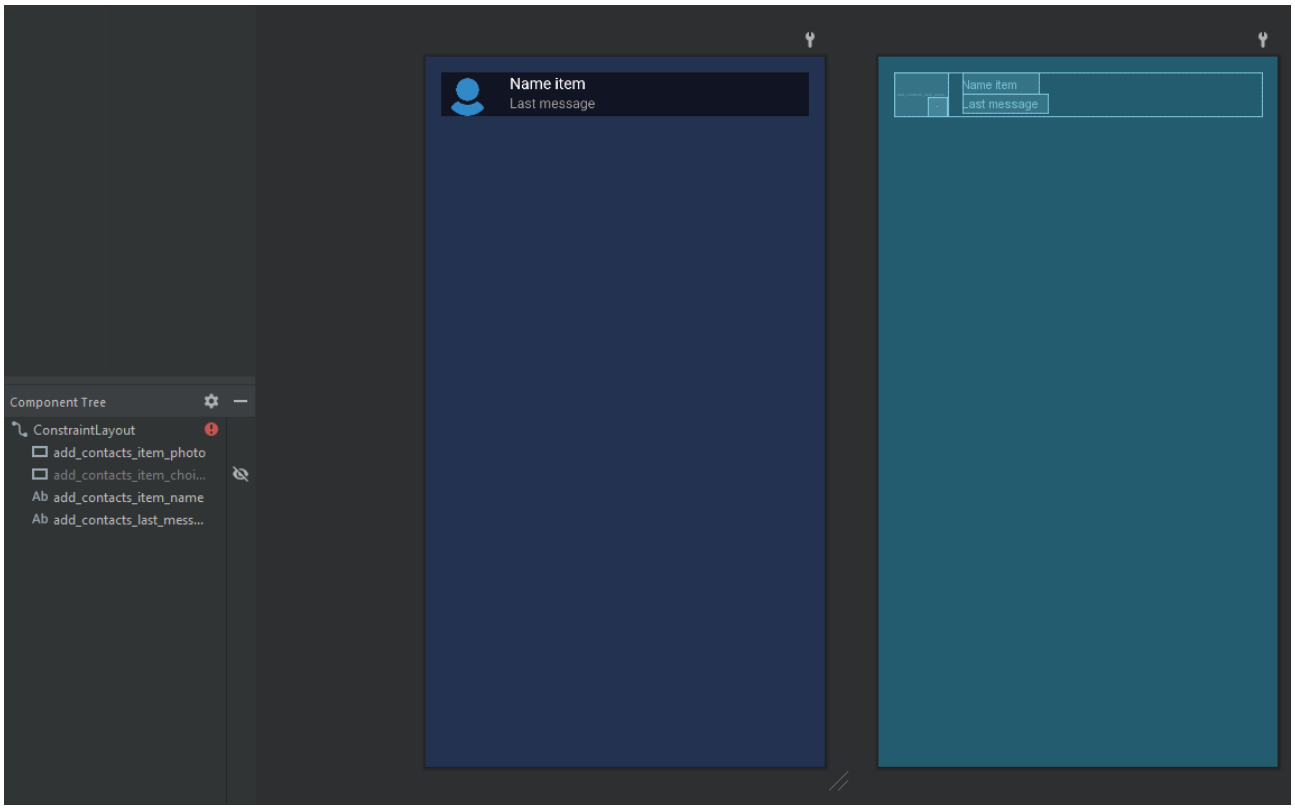

Рисунок 3.9 – Об’єкт для додавання контакта

• **Список контактів (Fragment contacts)** (рис. 3.10). Активність використовує шаблон форматування FrameLayout. Використовує елемент RecyclerView для роботи з списками контактів. Вона відповідає за виведення списку контактів, які автоматично зчитуються з телефону.

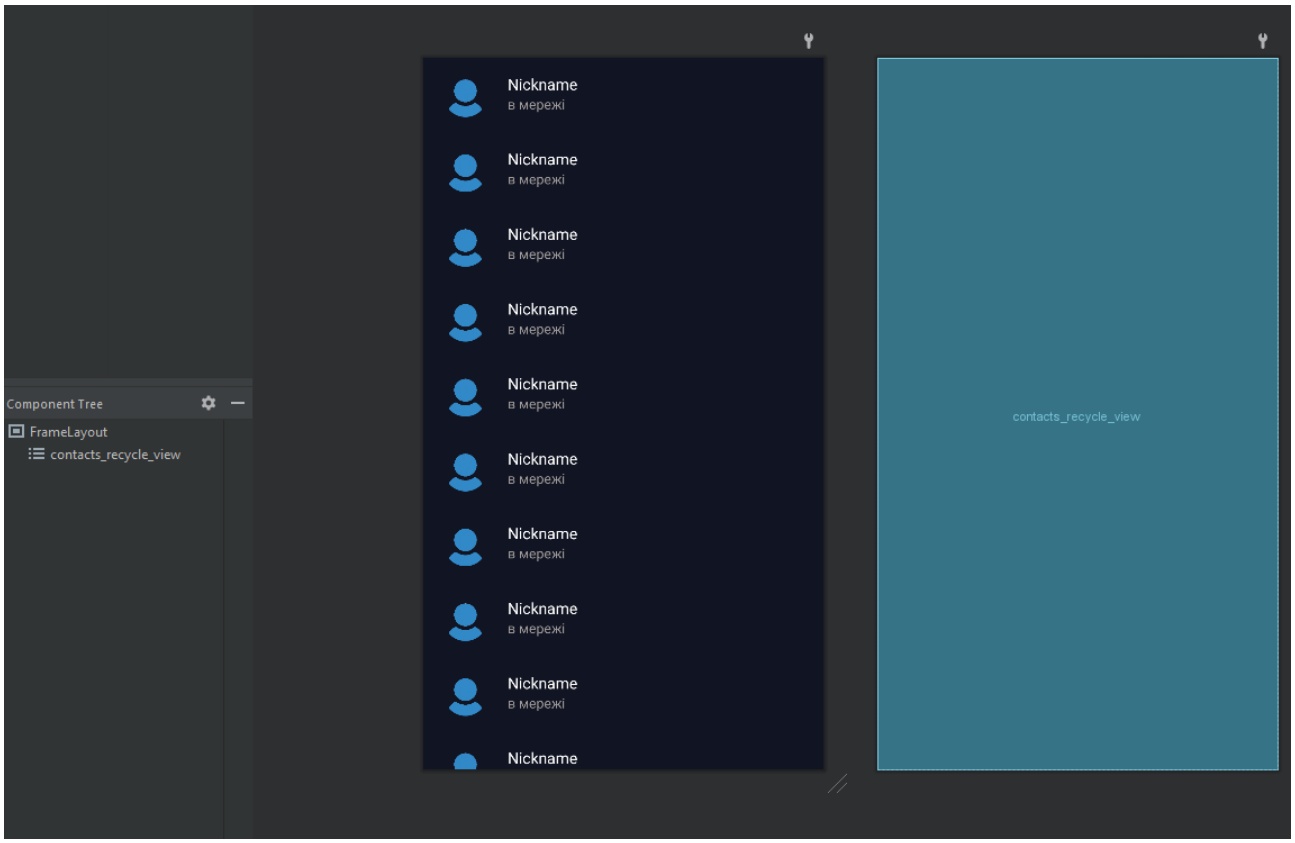

Рисунок 3.10 – Список контактів

• **Макет налаштувань (Fragment settings)** (рис. 3.11). Активність використовує шаблон форматування ConstraintLayout. Складається з 4 блоків:

– блок з інформацією про користувача (фото, ім’я, активність), кнопка для зміни аватарки;

– блок з номером телефона (є можливість його змінити);

– блок з Ім’ям користувача (є можливість його змінити);

– блок з інформацією про себе (є можливість її змінити).

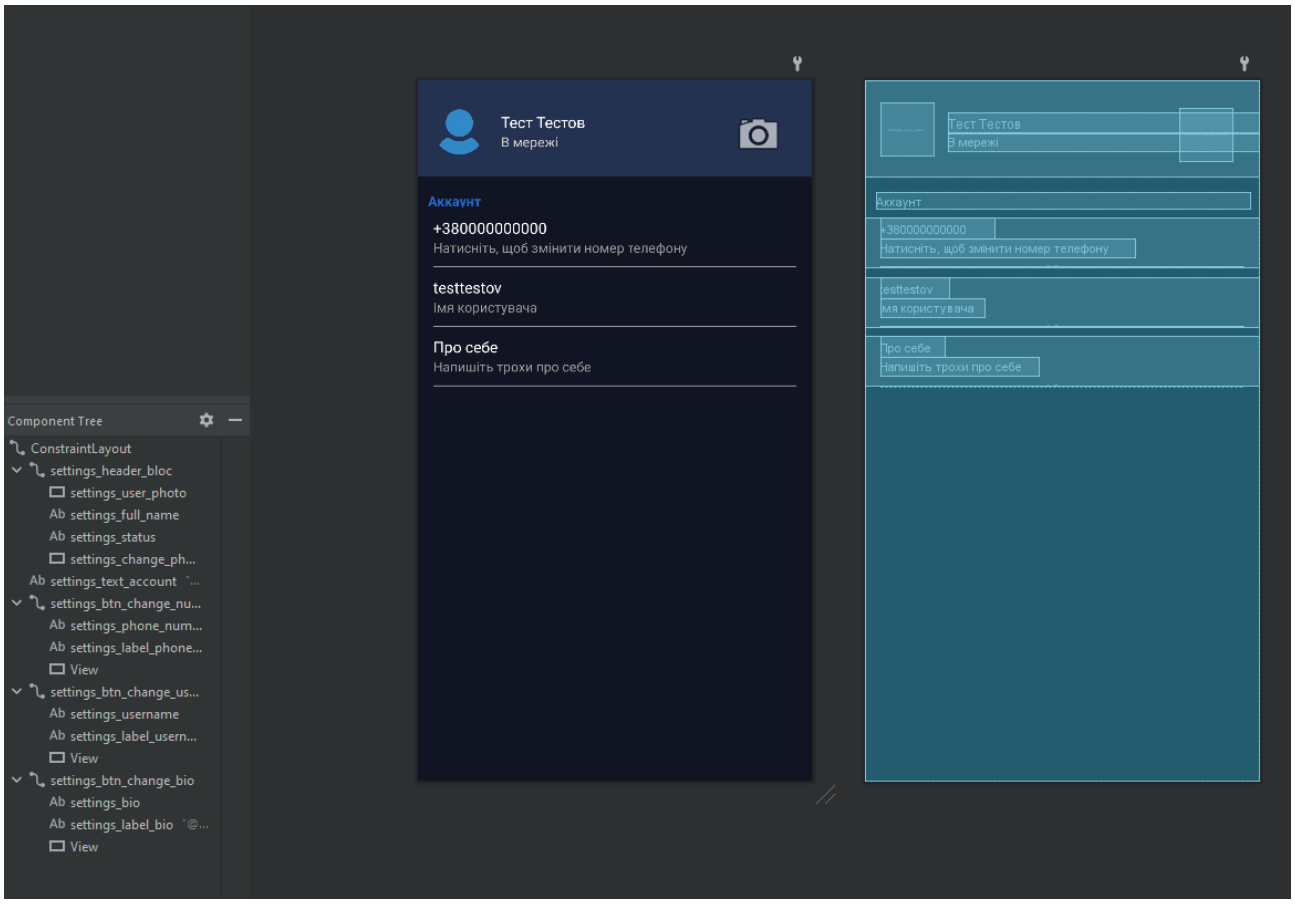

Рисунок 3.11 – Макет налаштувань

• **Активність для зміни імені і прізвища (Fragment change name)** (рис. 3.12). Активність використовує шаблон форматування ConstraintLayout. Складається з двох рядків для введення нового імені та прізвища.

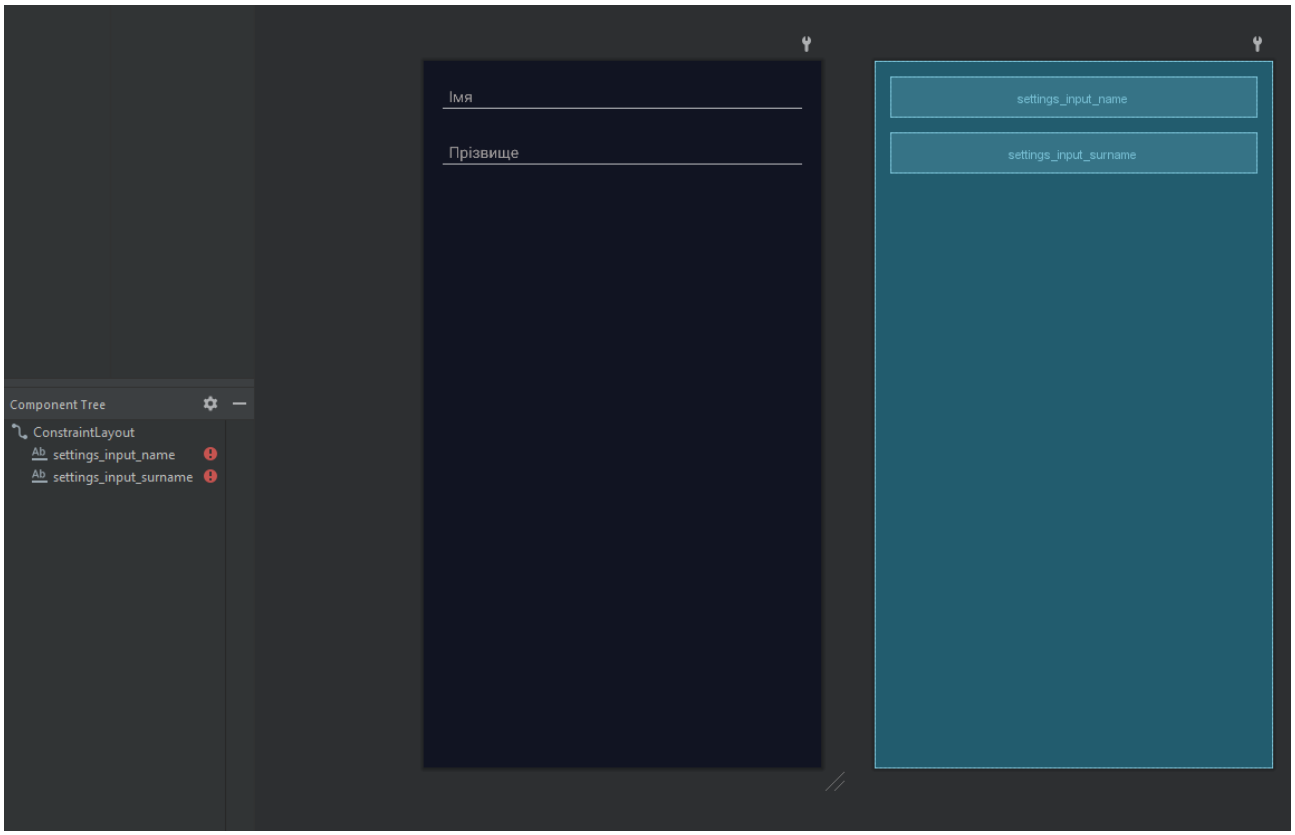

Рисунок 3.12 – Активність для зміни імені і прізвища

• **Активність для зміни імені користувача (Fragment change username)** (рис. 3.13). Активність використовує шаблон форматування ConstraintLayout. Складається з рядка для введення нового імені.

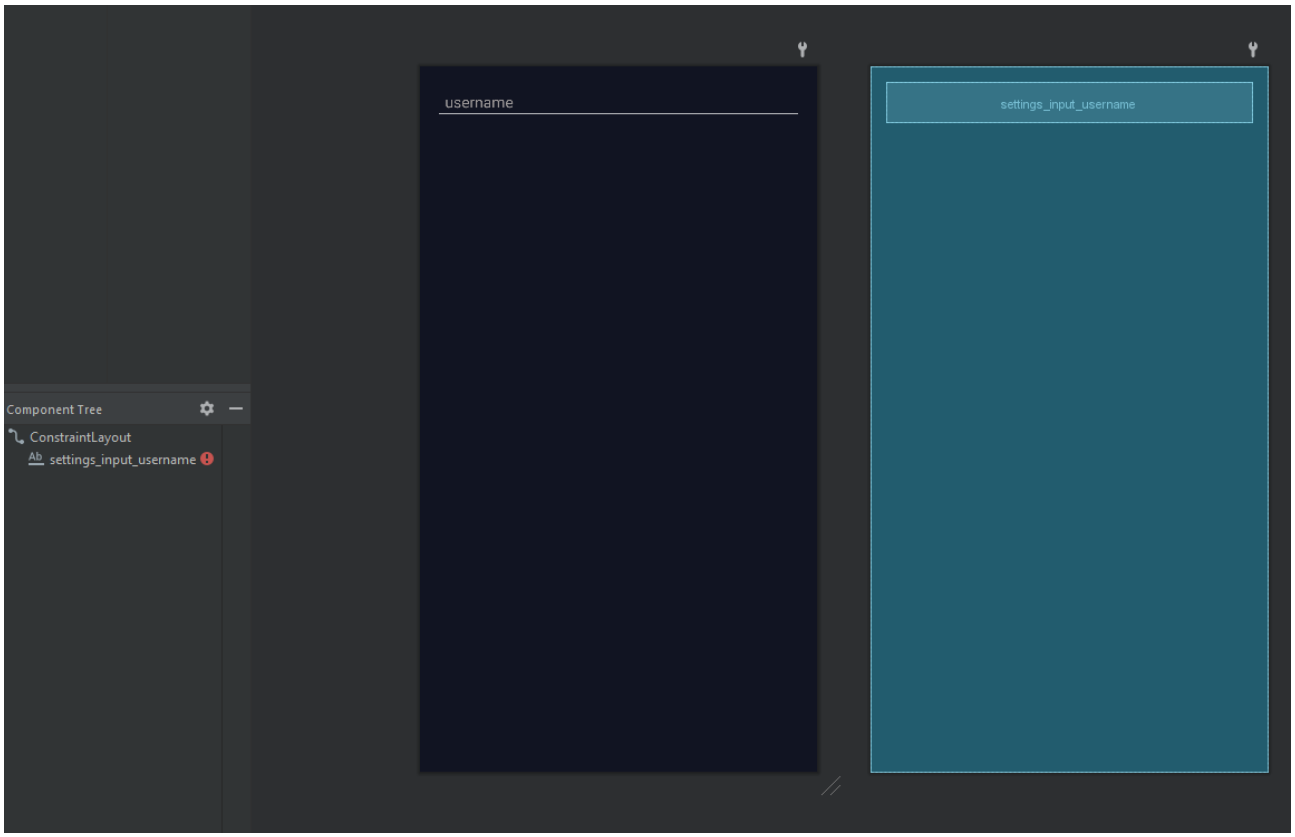

Рисунок 3.13 – Активність для зміни імені користувача

• **Активність для зміни опису (Fragment change bio)** (рис. 3.14). Активність використовує шаблон форматування ConstraintLayout. Складається з рядка для введення нової інформації користувача.

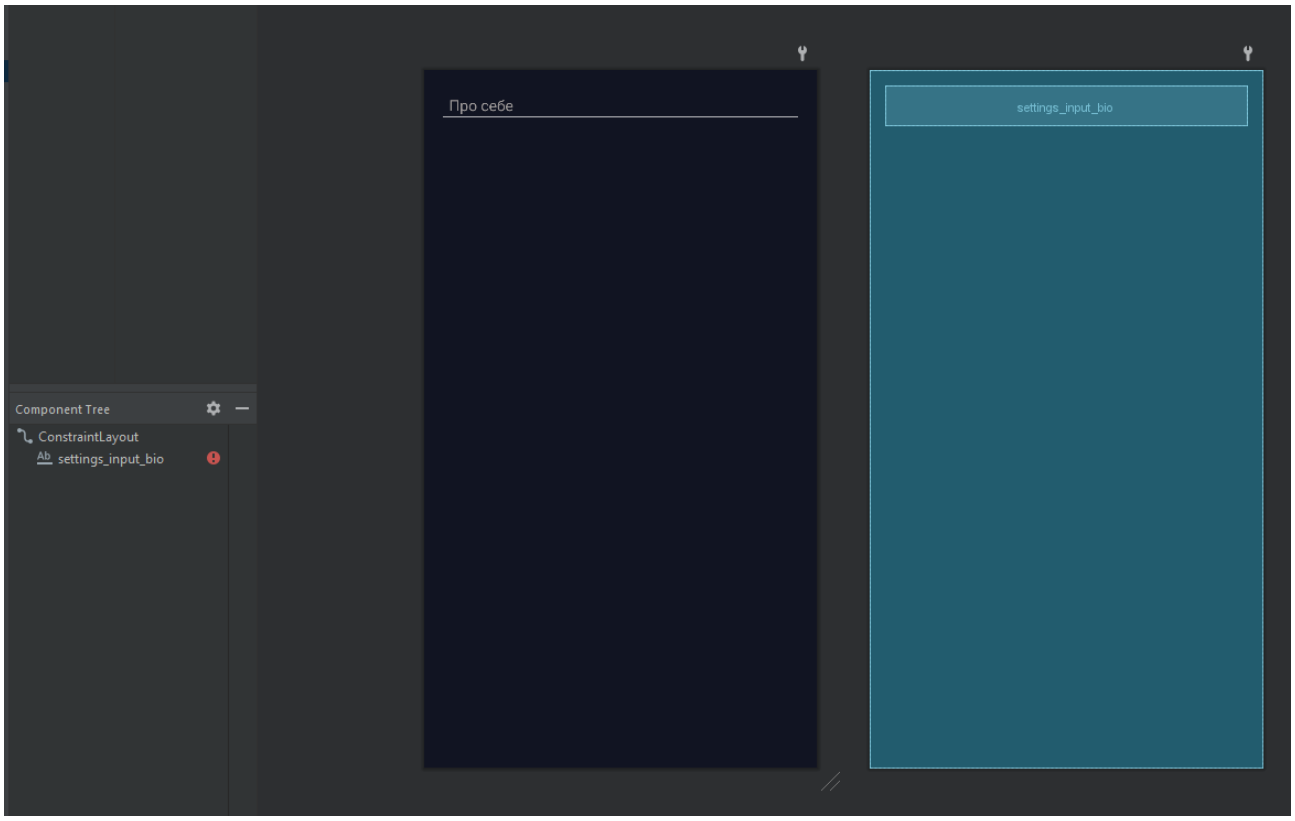

Рисунок 3.14 – Активність для зміни опису

• **Активність створення груп (Fragment create group)** (рис. 3.15). Активність використовує шаблон форматування ConstraintLayout. Складається з Кнопки для вибору аватарки для групи, рядка для введення назви групи, лічильника кількості запрошених учасників групи, список контактів, які можна додати в групу і кнопка для створення групи.

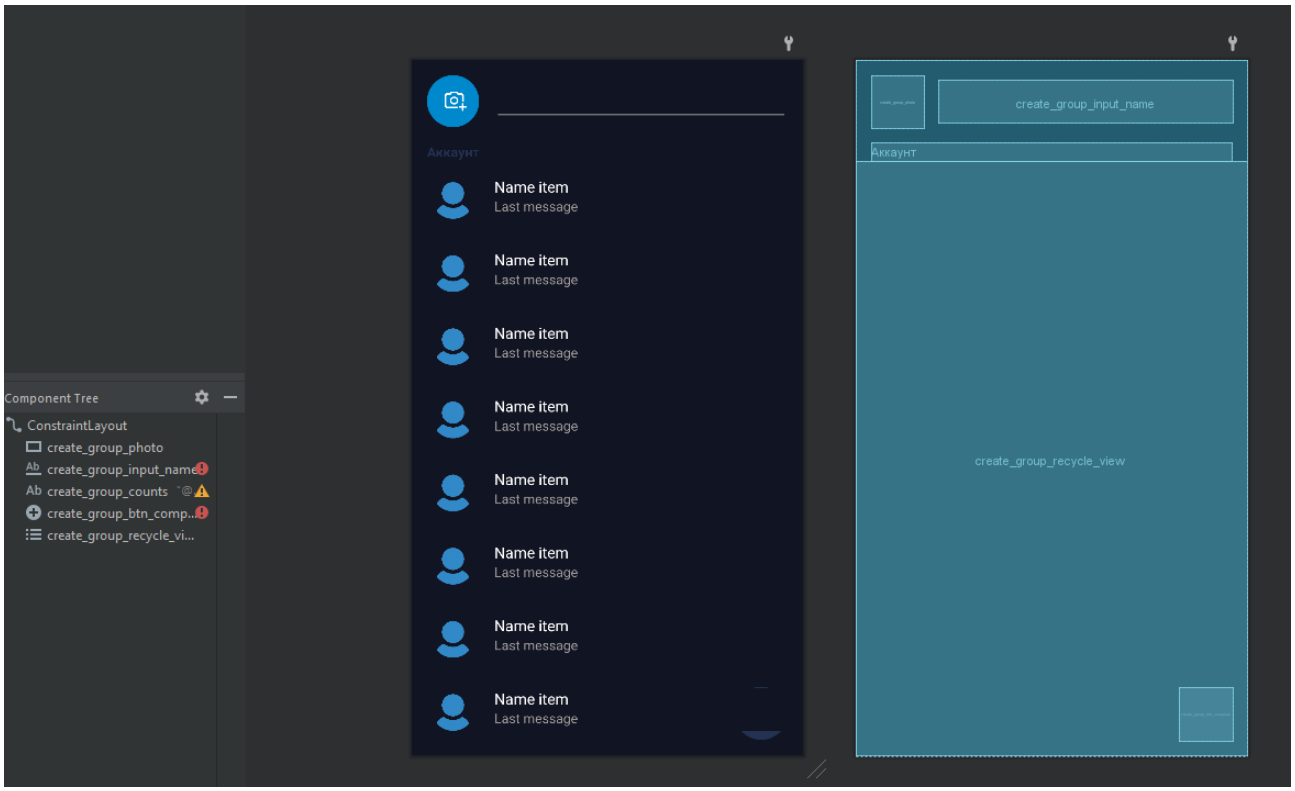

Рисунок 3.15 – Активність створення груп

• **Макет чату (Fragment single chat)** (рис. 3.16). Активність використовує шаблон форматування CoordinatorLayout. Складається з області для відображення повідомленнь, рядку для введення повідомлення, кнопки для відправлення голосових повідомлень, кнопки для відправлення файлів і активності Choise upload для вибору типу вівдвантаженого файлу.

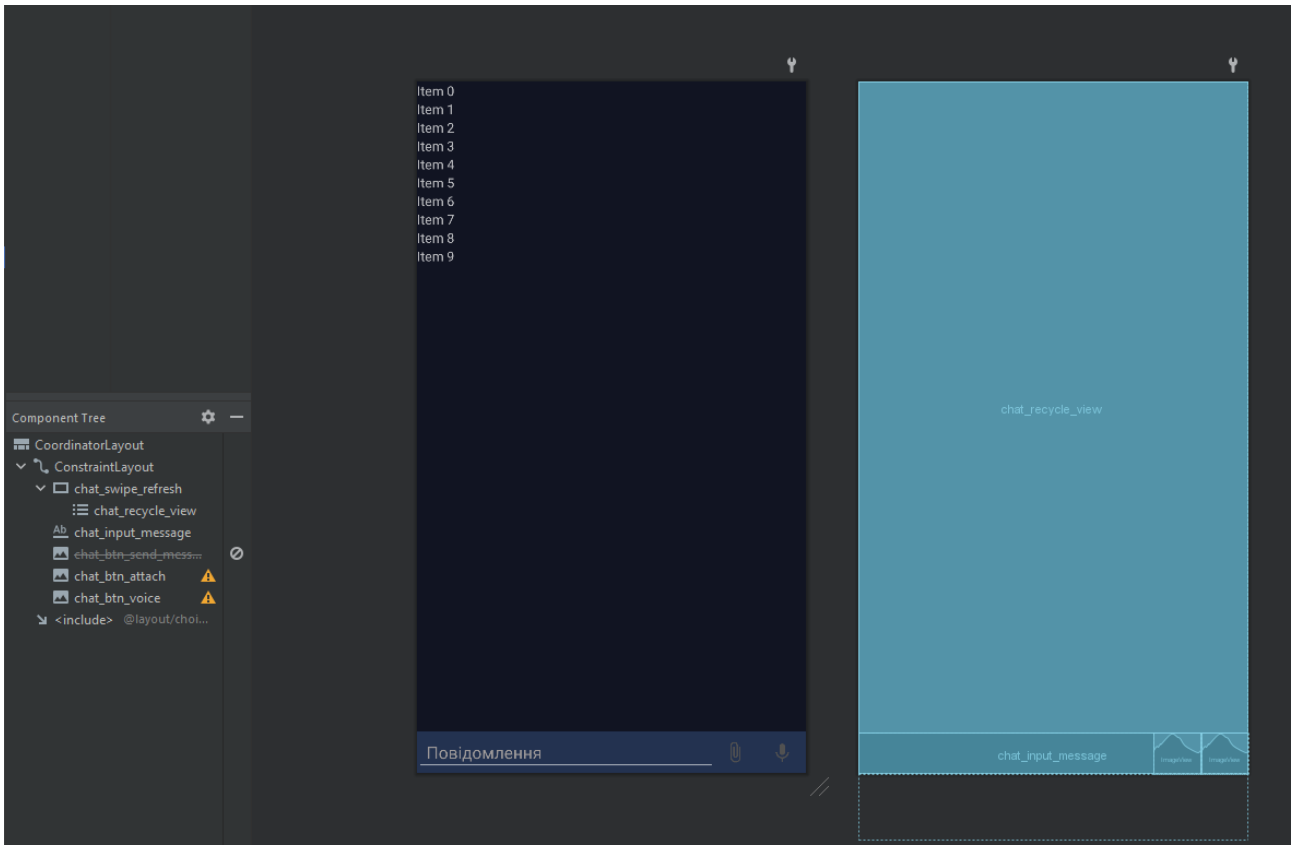

Рисунок 3.16 – Макет чату

• **Активність вибору типу файла (Choise upload)** (рис. 3.17). Активність для вибору типу файлу, який буде відвантажуватись.

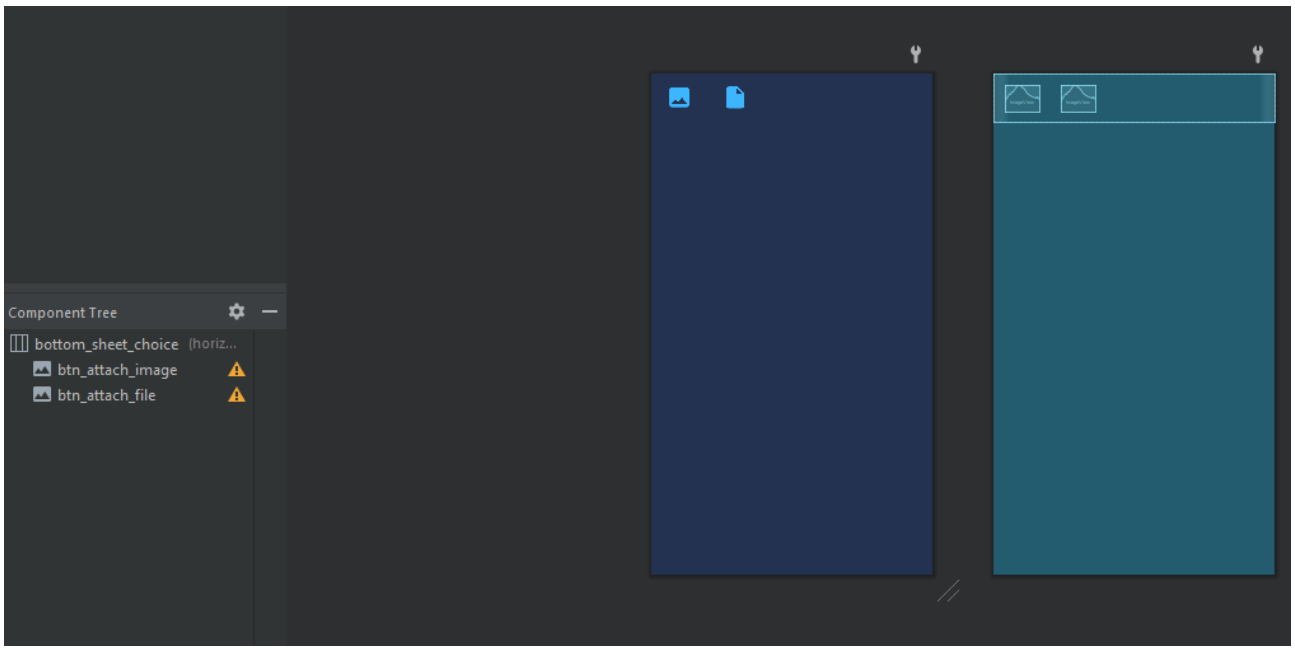

Рисунок 3.17 – Активність вибору типу файла

• **Макет вигляду відправленого і отриманого в чаті файла (Message item file)** (рис. 3.18). Макет відповідає за відображення файлів в чаті.

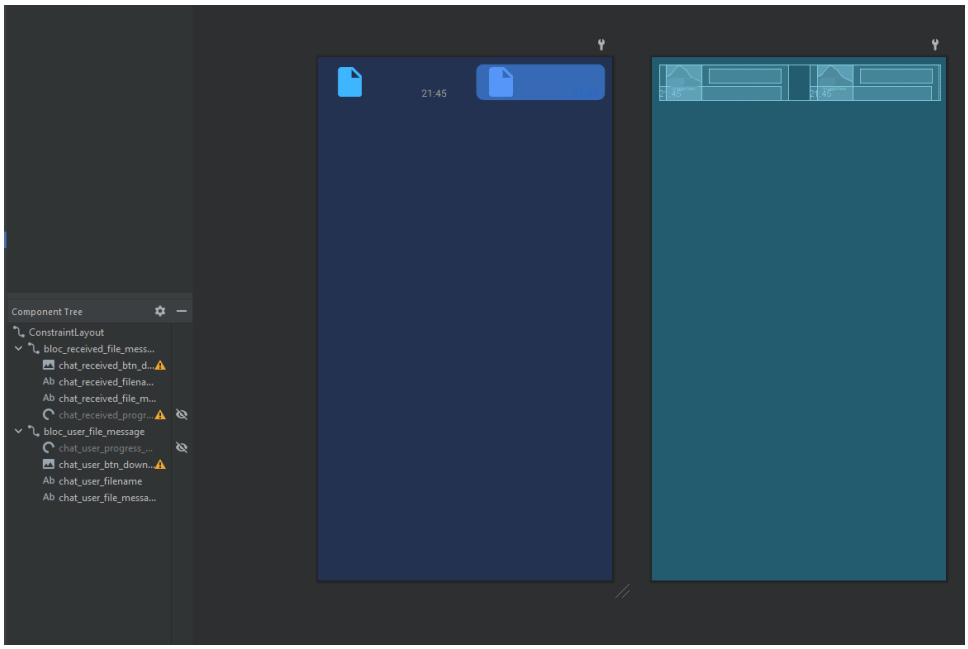

Рисунок 3.18 – Макет вигляду відправленого і отриманого в чаті файла

• **Макет вигляду відправленого і отриманого в чаті картинки (Message item image)** (рис. 3.19). Макет відповідає за відображення картинки в чаті.

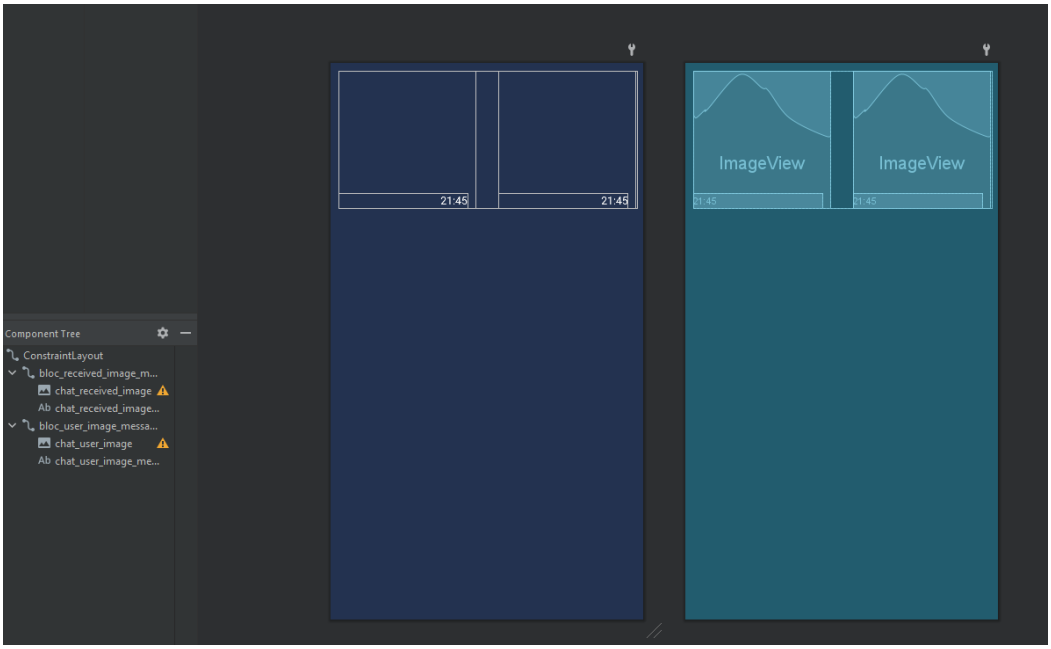

Рисунок 3.19 – Макет вигляду відправленого і отриманого в чаті картинки

• **Макет вигляду відправленого і отриманого в чаті тексту (Message item text)** (рис. 3.20). Макет відповідає за відображення тексту в чаті.

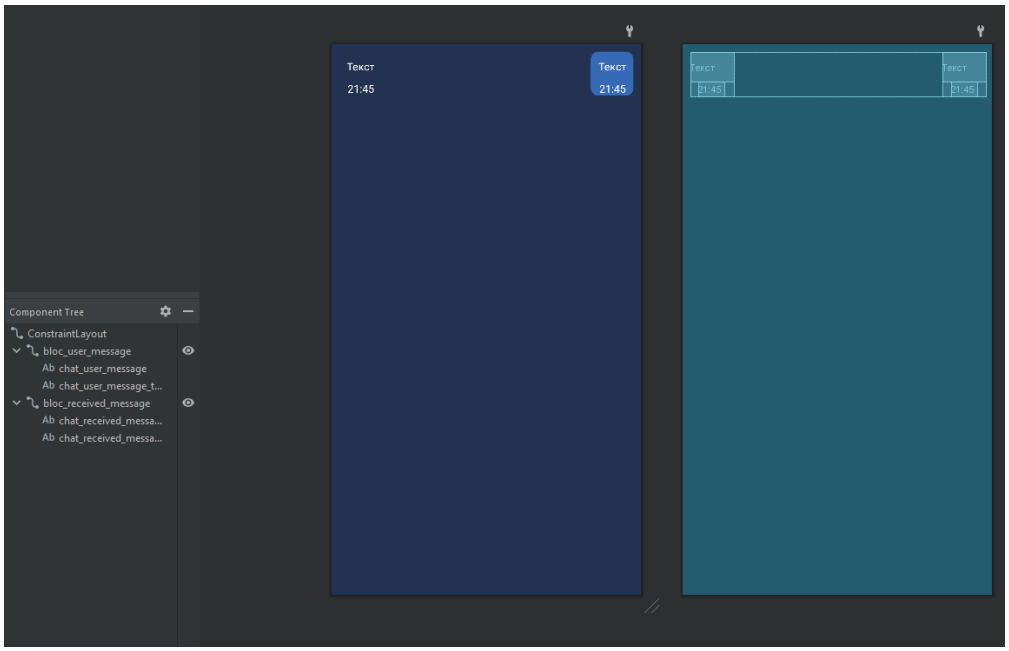

Рисунок 3.20 – Макет вигляду відправленого і отриманого в чаті тексту

• **Макет вигляду відправленого і отриманого в чаті голосового повідомлення (Message item voice)** (рис. 3.21). Макет відповідає за відображення голосового повідомлення в чаті.

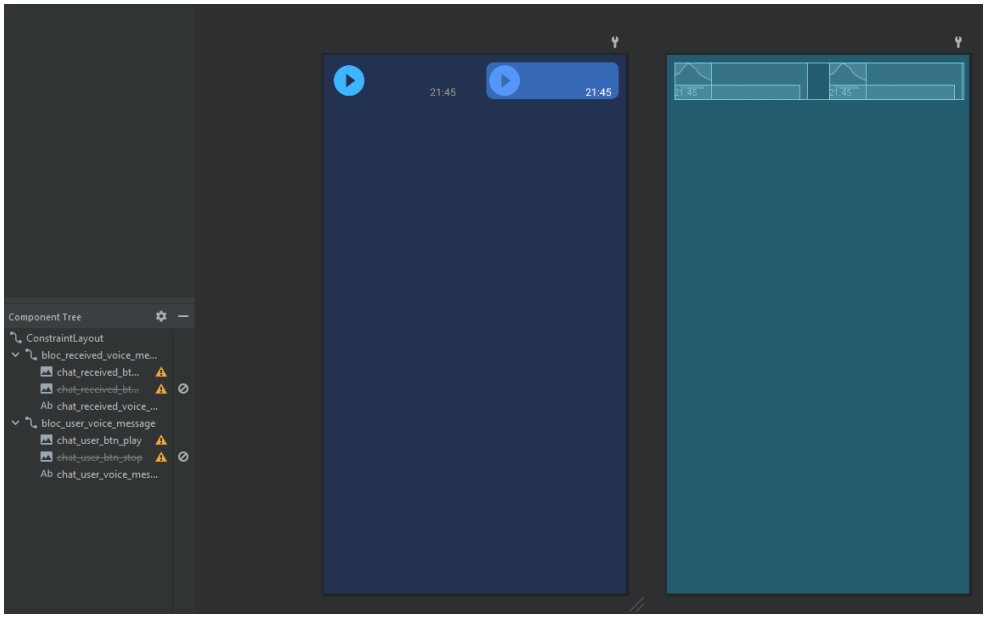

Рисунок 3.21 – Макет вигляду відправленого і отриманого в чаті голосового повідомлення

3.3 Опис функціоналу

При запуску програми на екран відображається вікно авторизації, де потрібно ввести номер телефона (рис. 3.22).

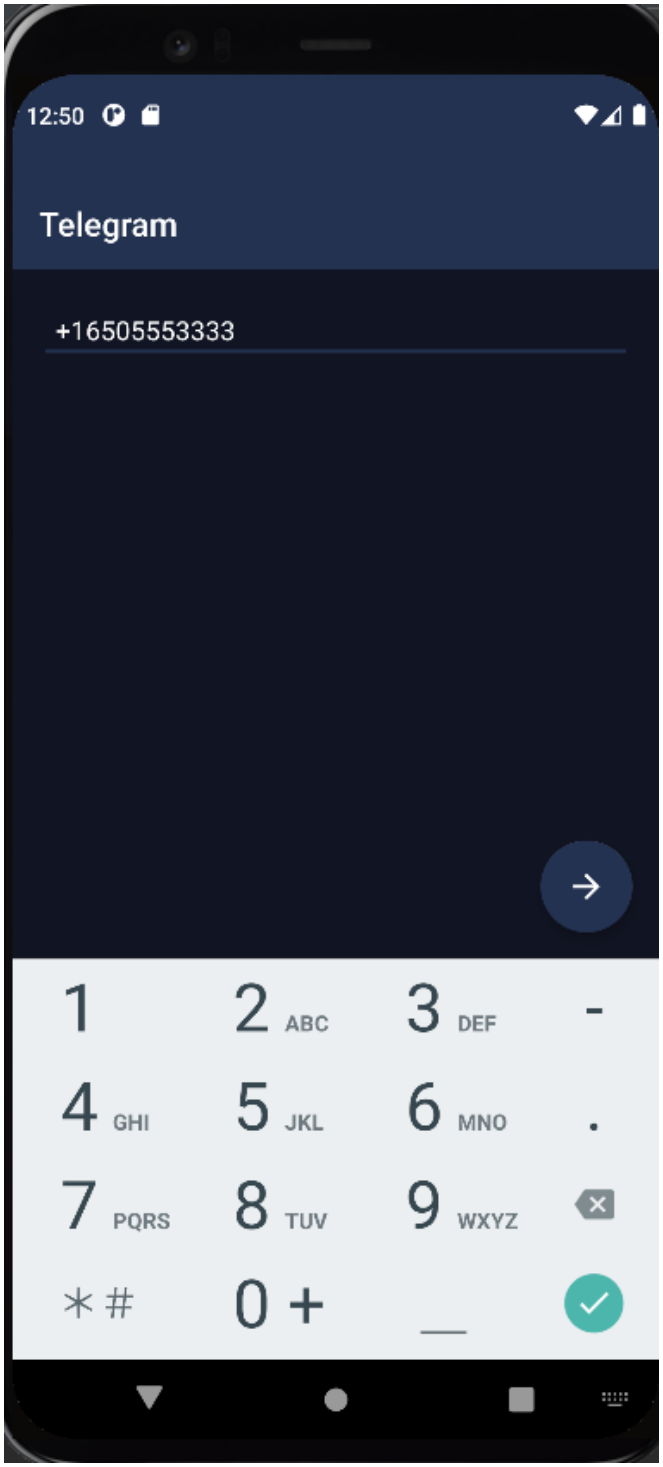

Рисунок 3.22 – Вікно введення номера телефона

Після введення номера потрібно натиснути на кнопку для того, щоб на ваш номер відправилась смс з кодом перевірки і відкрилось вікно для його введення (рис. 3.23).

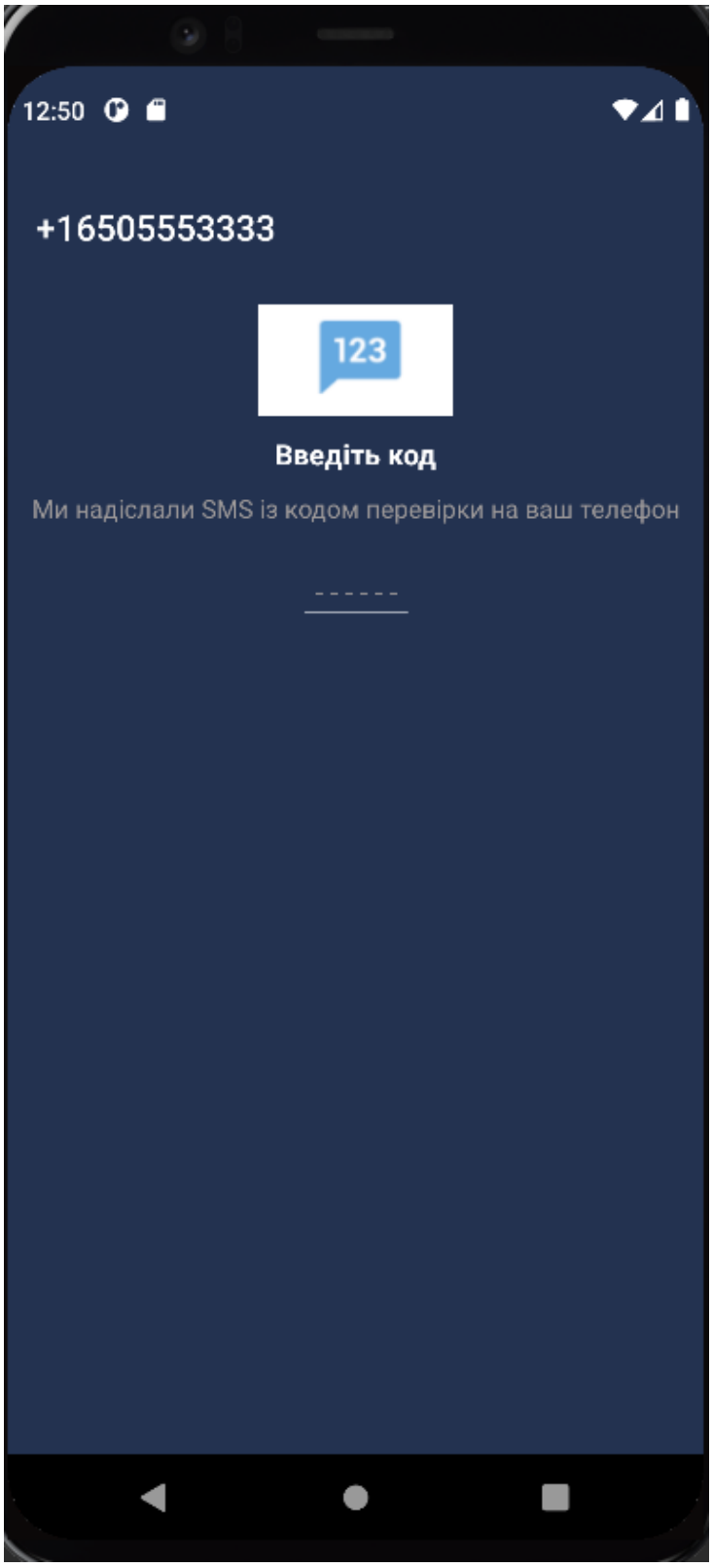

Рисунок 3.23 – Вікно введення коду

Після введення коду відкривається вікно, зображене на рис. 3.24, де є головне меню, аватарка, ім’я та номер користувача, і на задньому фоні список останніх чатів.

Для виклику головного меню з вибором категорій потрібно свайпнути зліва направо.

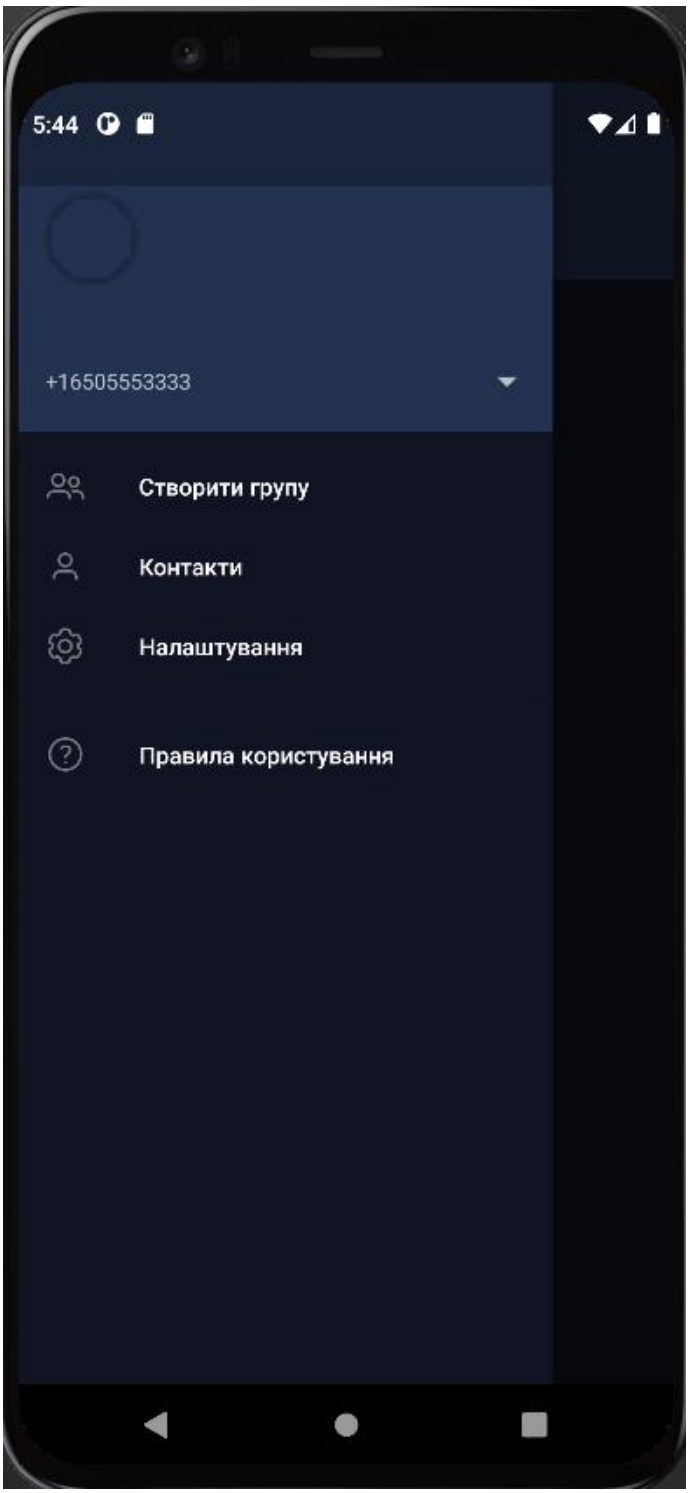

Рисунок 3.24 – Головне меню

Після першого входу в застосунок потрібно зайти в налаштування і змінити фото, ім’я і додати інформацію про себе. Натиснувши налаштування відкривається меню налаштувань (рис. 3.25). Для зміни імені потрібно натиснути на Ім’я користувача (виділено червоним на рис. 3.25), ввести нове ім’я і натиснути на галочку для збереження.

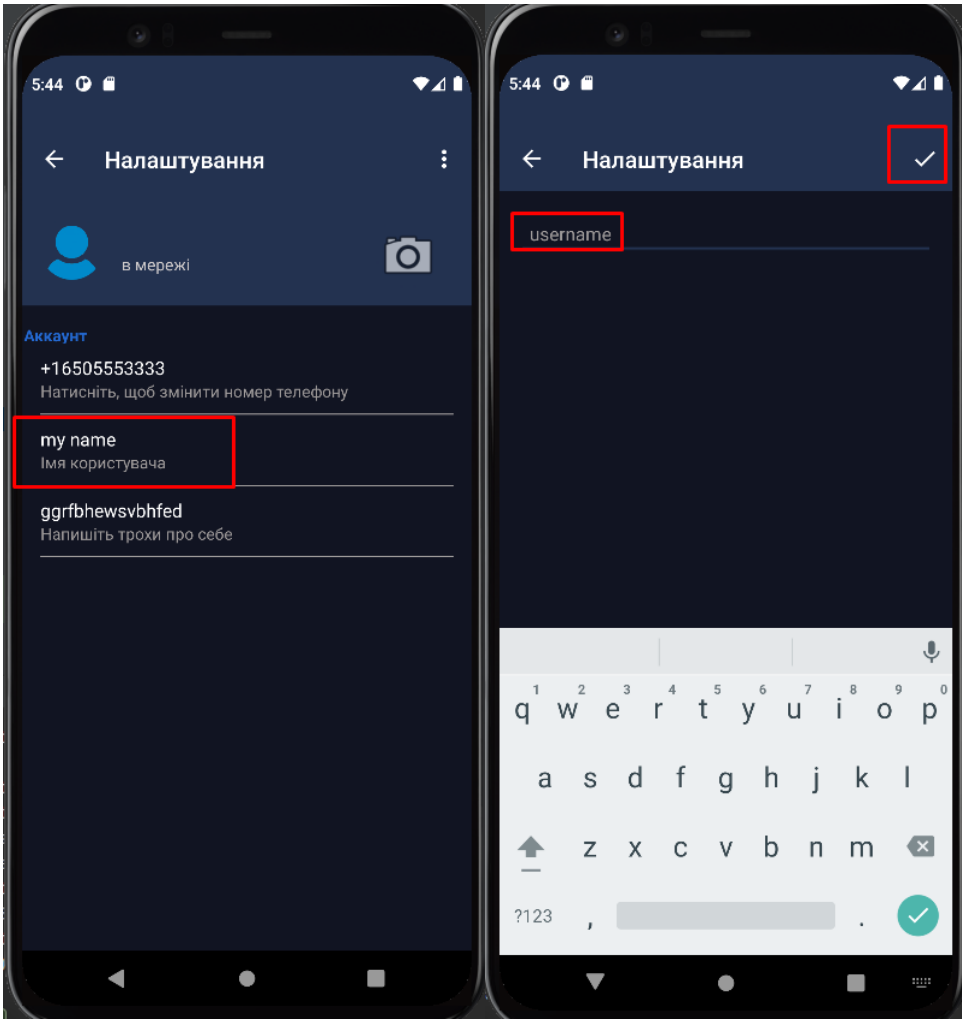

Рисунок 3.25 – Вигляд меню налаштувань

Для зміни опису потрібно натиснути на “Напишіть трохи про себе” (виділено червоним на рис. 3.26), ввести новий опис і натиснути на галочку для збереження.

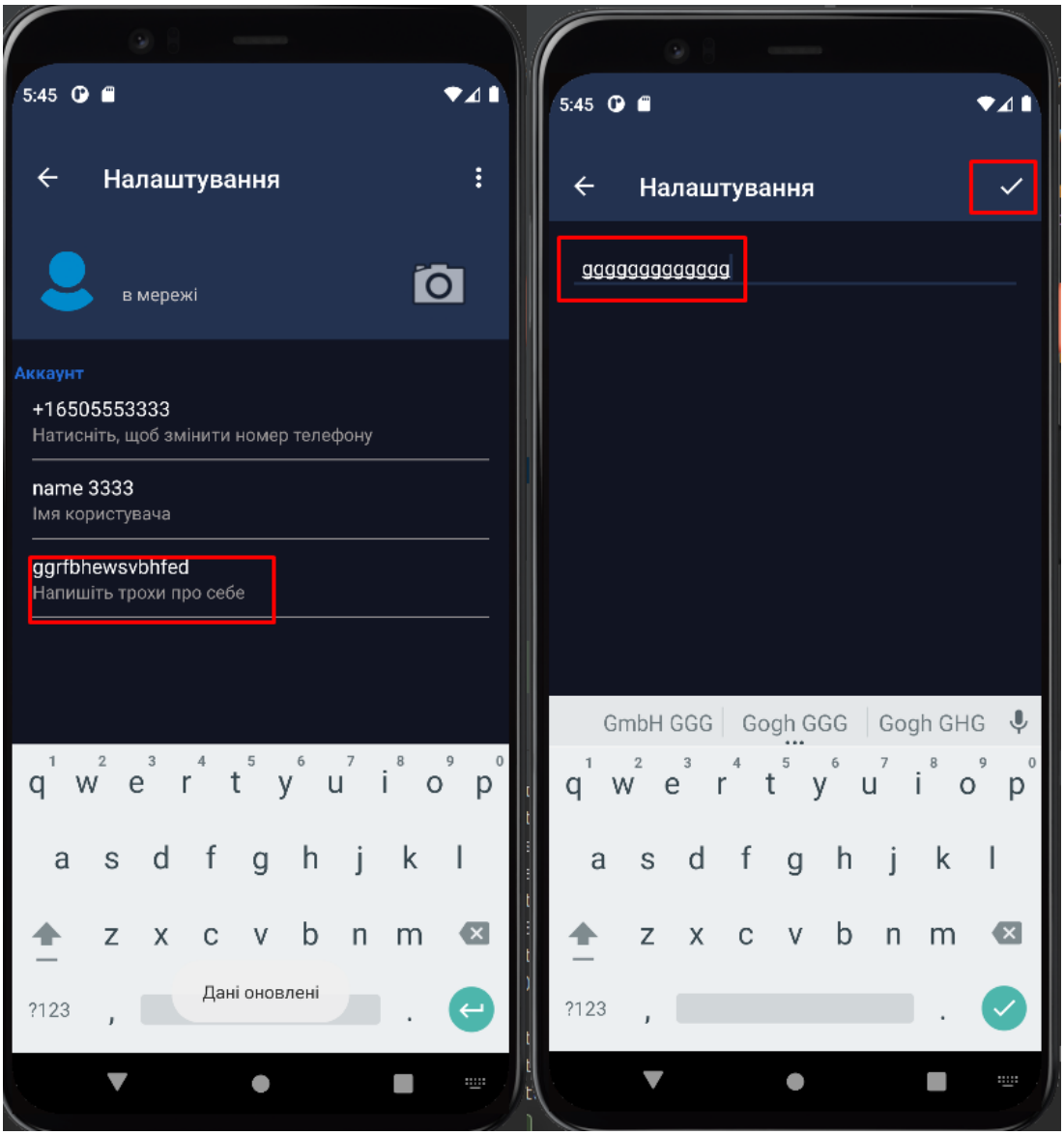

Рисунок 3.26 – Зміна опису

Для зміни аватарки потрібно натиснути на кнопки у вигляді фотоапарата (виділено червоним на рис. 3.27), вибрати місцезнаходження фото, вибрати фото, виділити область, яка буде використовуватись і натиснути “Crop” після чого аватарка користувача буде змінена (рис. 3.28).

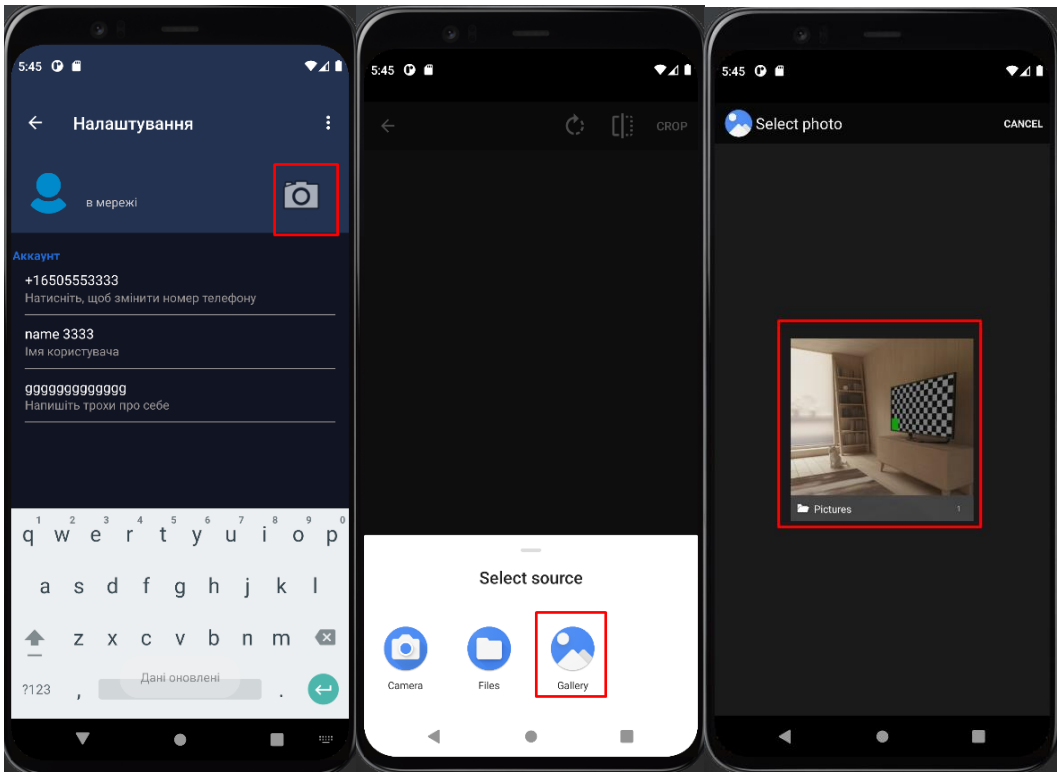

Рисунок 3.27 – Зміна аватарки

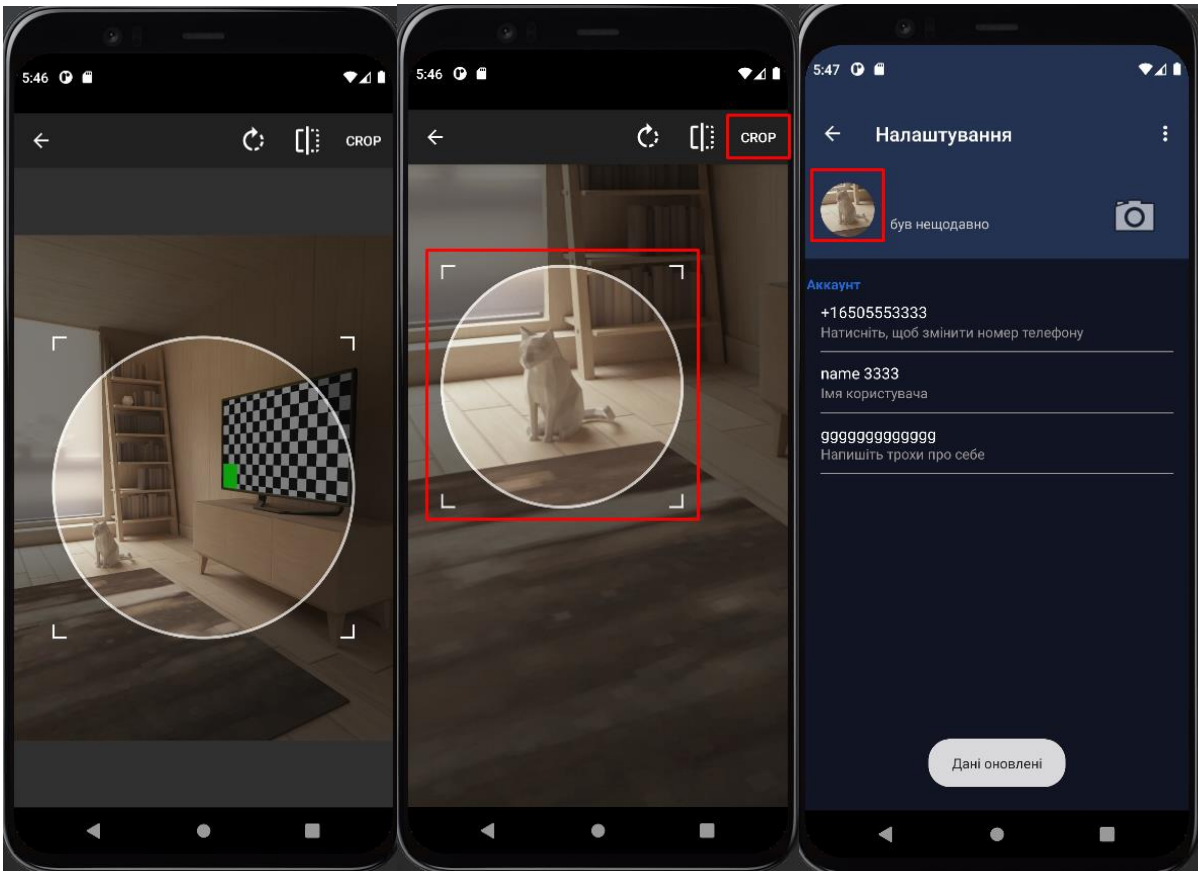

Рисунок 3.28 – Зміна аватарки

Також можна змінити ім’я і прізвище натиснувши на три крапки, “Змінити ім'я” і після введення натиснувши галочку (рис. 3.29).

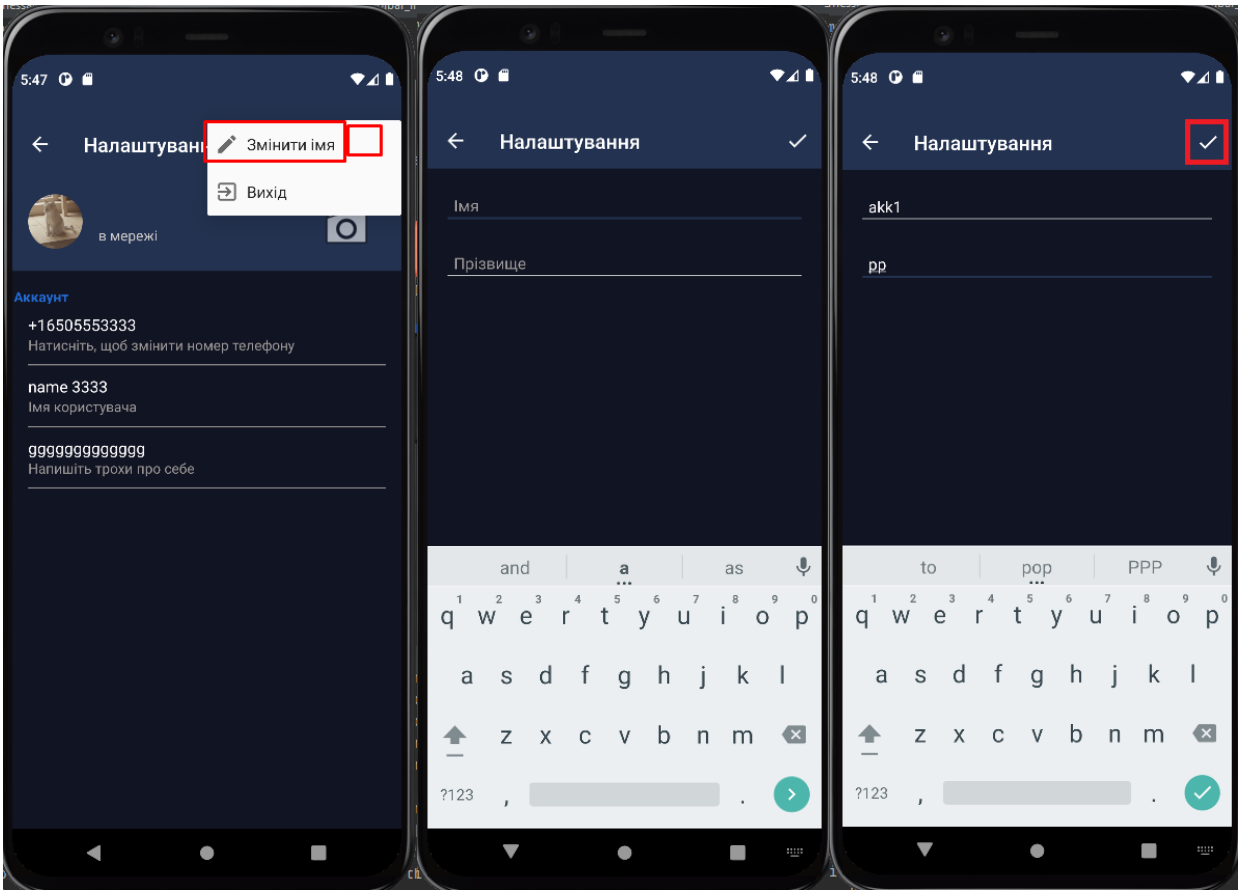

Рисунок 3.29 – Зміна імені і прізвища

Після попередніх дій аккаунт налаштований (рис. 3.30).

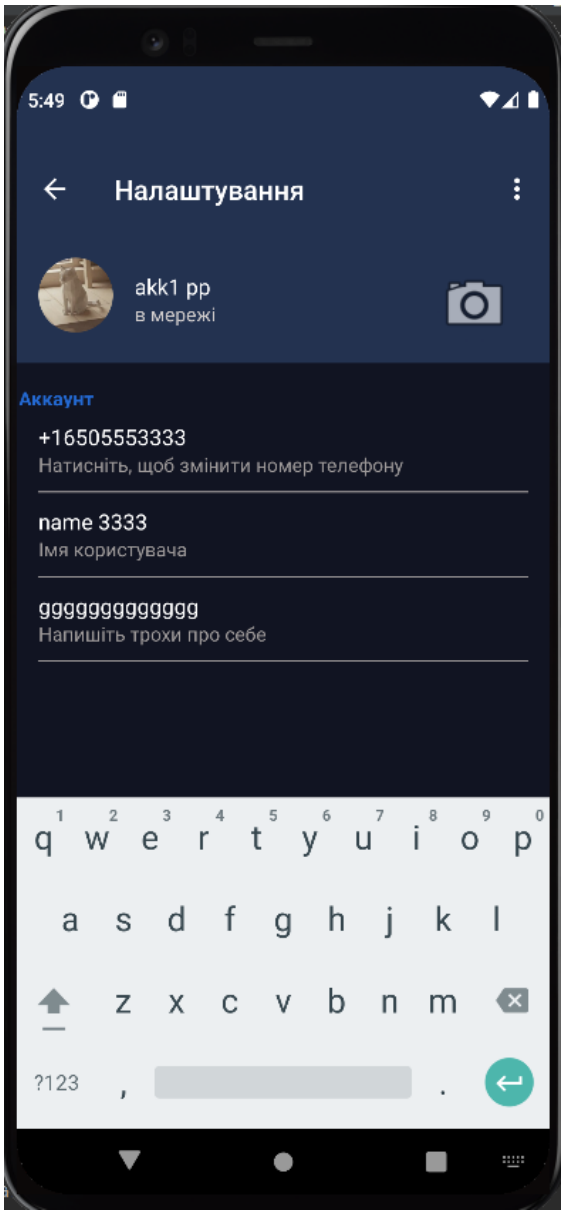

Рисунок 3.30 – Аккаунт після налаштувань

Інформацію про застосунок і правила користування можна подивитись натуснувши на “Правила користування” в головному меню (рис. 3.31).

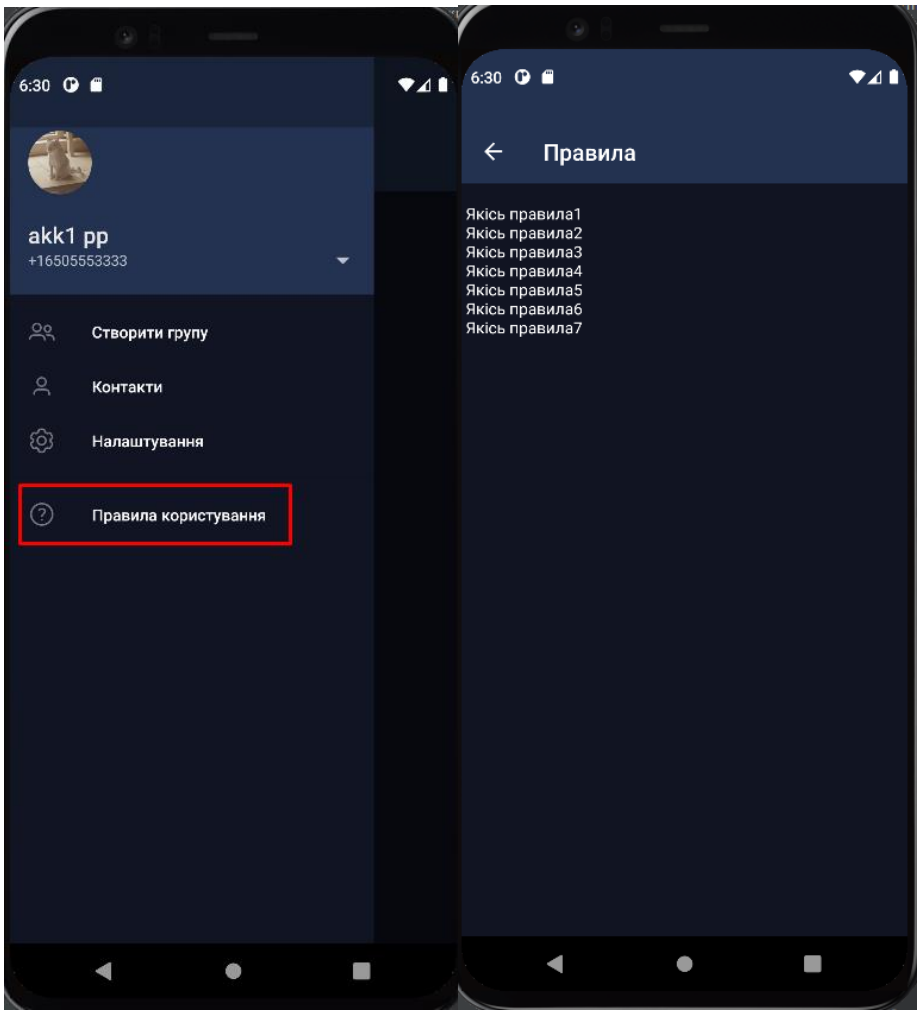

Рисунок 3.31 – Інформація/правила

Для створення групи потрібно викликати головне меню, натиснути на “Створити групу”, виділити учасників, яких потрібно додати в групу і натиснути на кнопку справа знизу. Після цього потрібно вибрати фото для групи і ввести назву. Також знизу будуть відображатися користувачі, які будуть додані в групу (рис. 3.32-3.33).

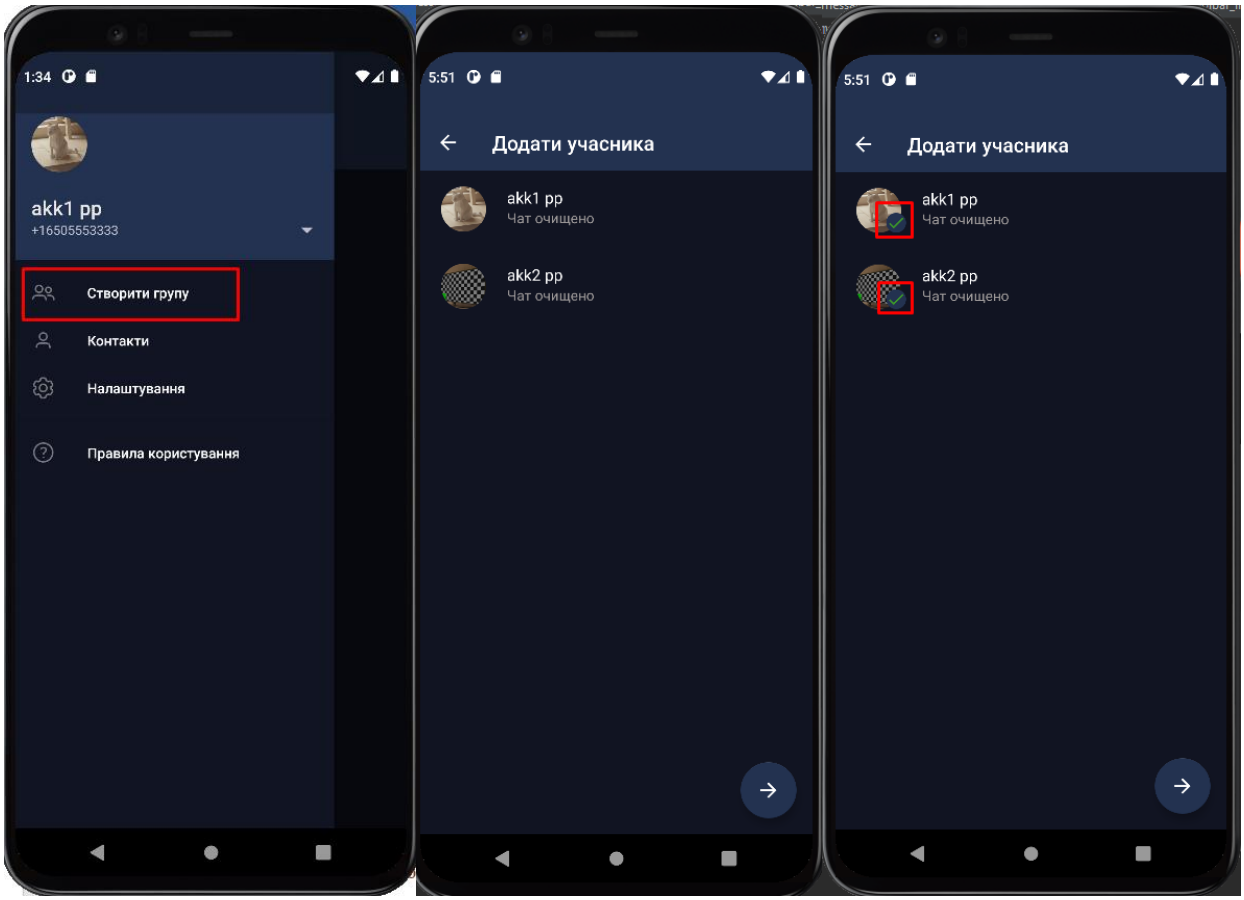

Рисунок 3.32 – Створення групи 1/2

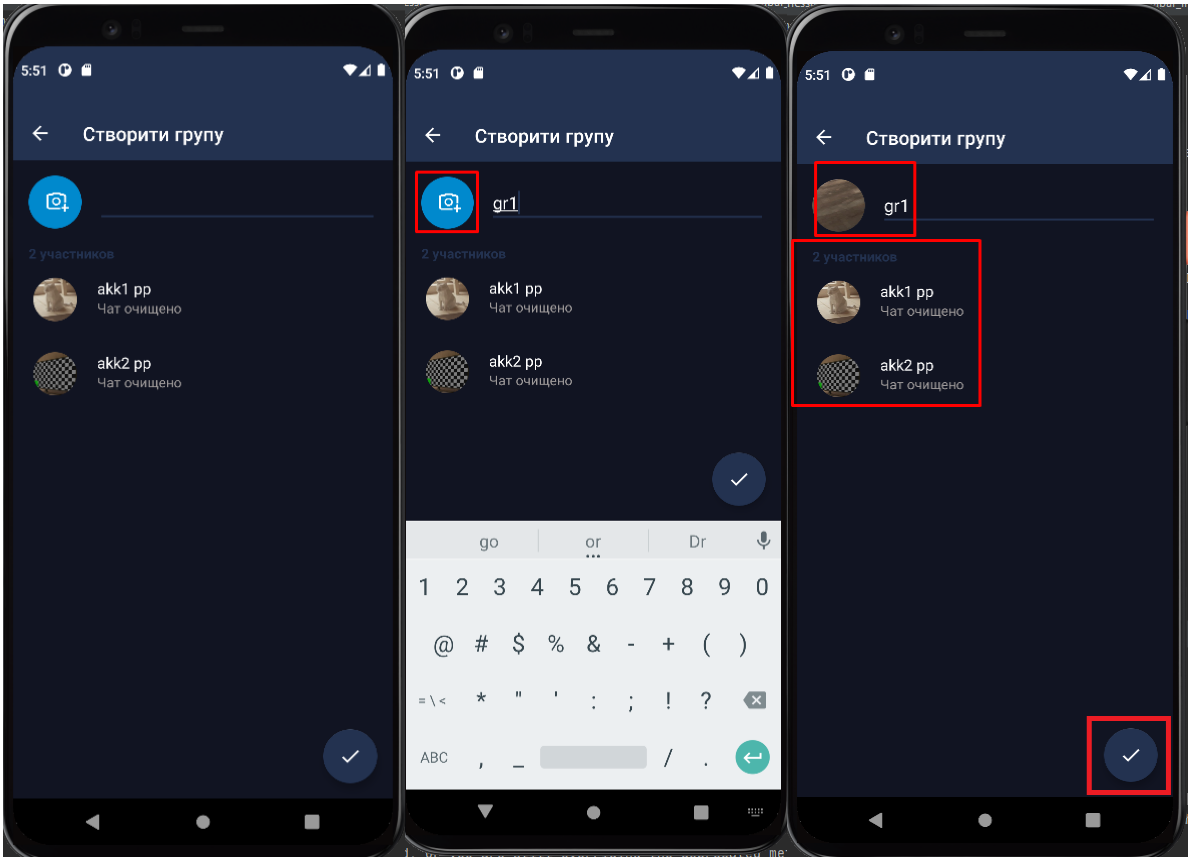

Рисунок 3.33 – Створення групи 2/2

Якщо на початковому вікні звернути головне меню, можна побачити список останніх чатів, де є назва групи/контакта і останнє повідомлення (рис. 3.34).

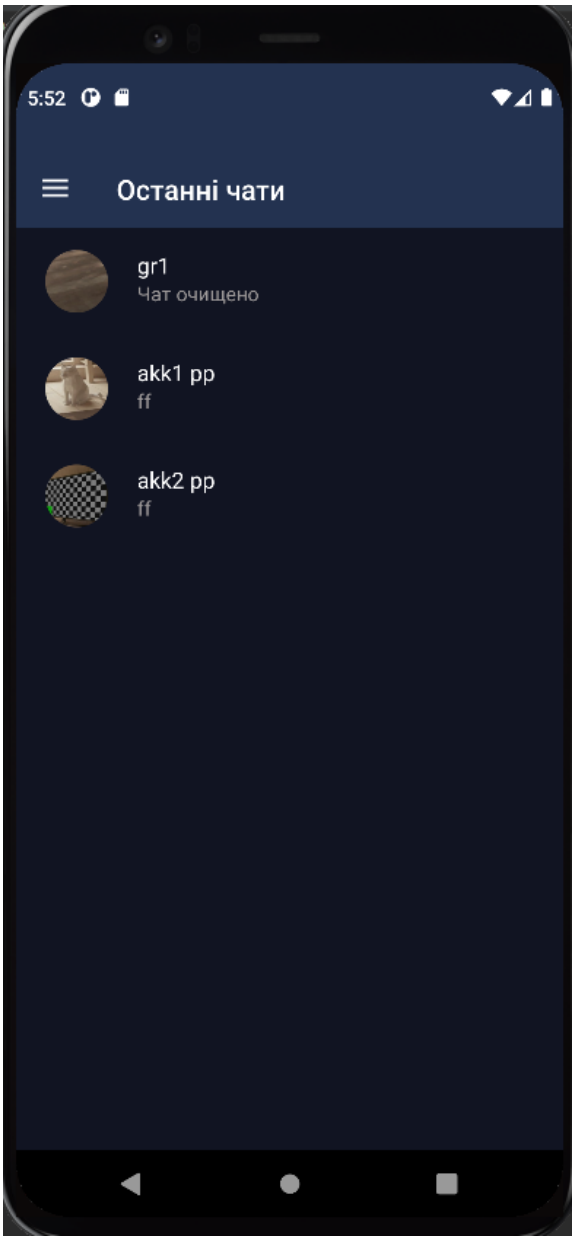

Рисунок 3.34 – Останні чати

Список контактів можна побачити натиснувши “Контакти” в головному меню (рис. 3.35).

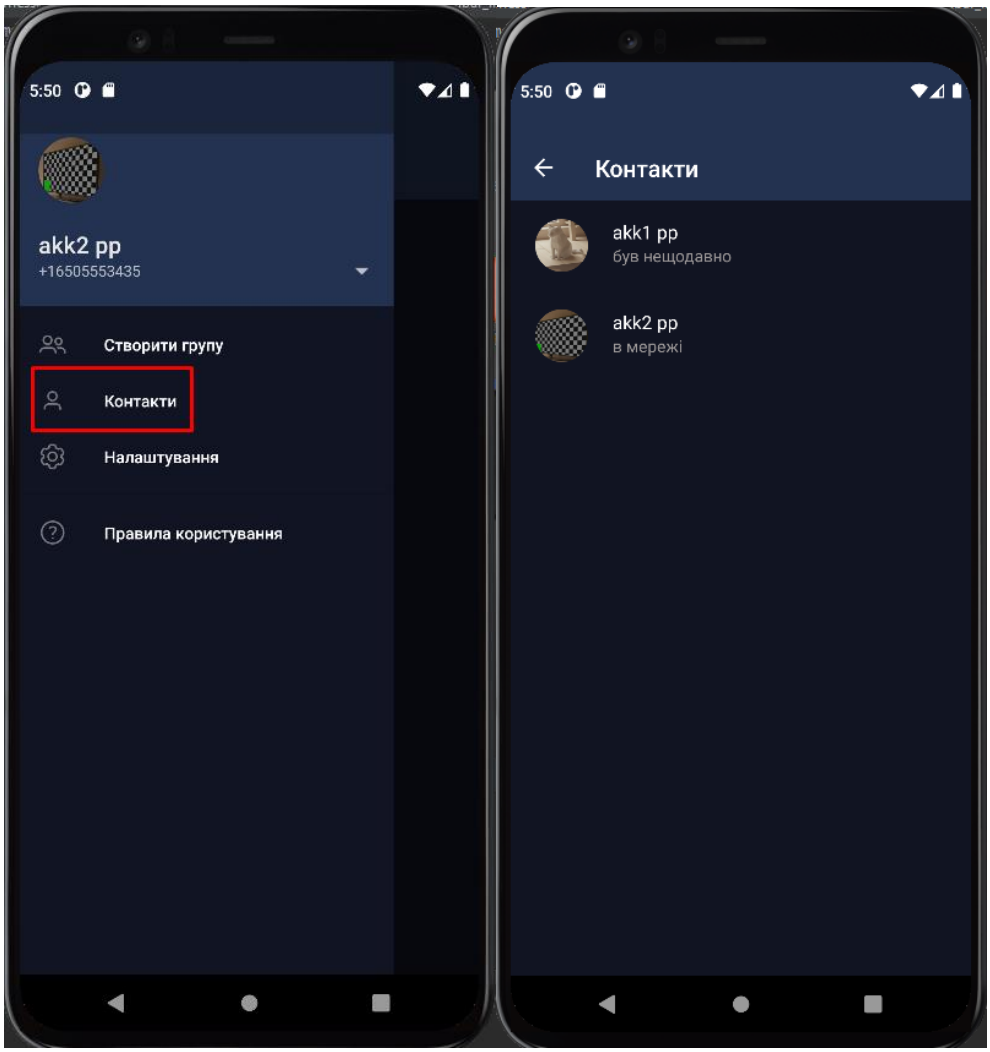

Рисунок 3.35 – Контакти

Найголовніше в мессенджері це чат, в чат можна відправити звичайний текст, ввівши його в рядок “Повідомлення”, відправити файл, натиснувши на кнопку прикріплення файла і вибравши його тип і вибрати сам файл, відправити голосове повідомлення затиснувши кнопку в вигляді мікрофону. Для легшого орієнтування між своїми та іншими повідомленнями вони розміщуються з різних країв і відрізняються за кольором (рис. 3.36-3.38).

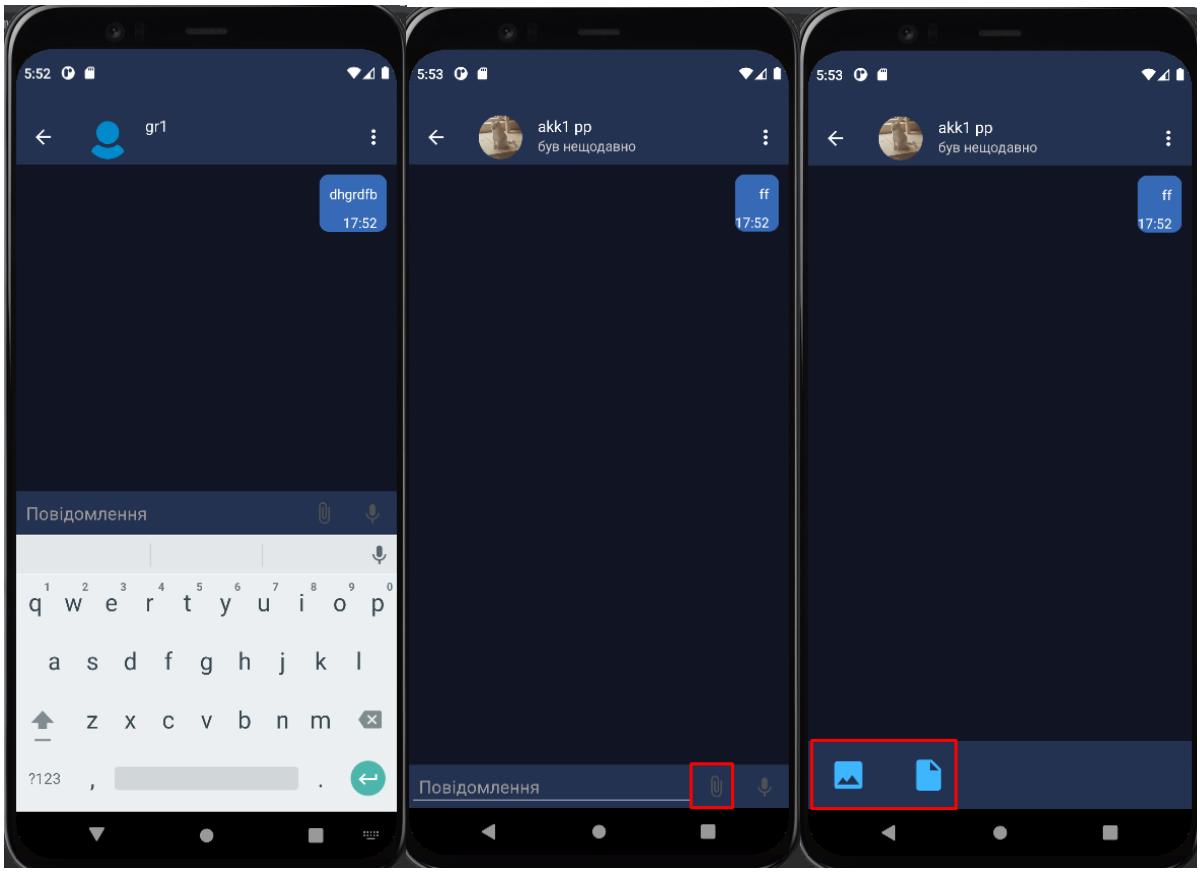

Рисунок 3.36 – Чати 1/3

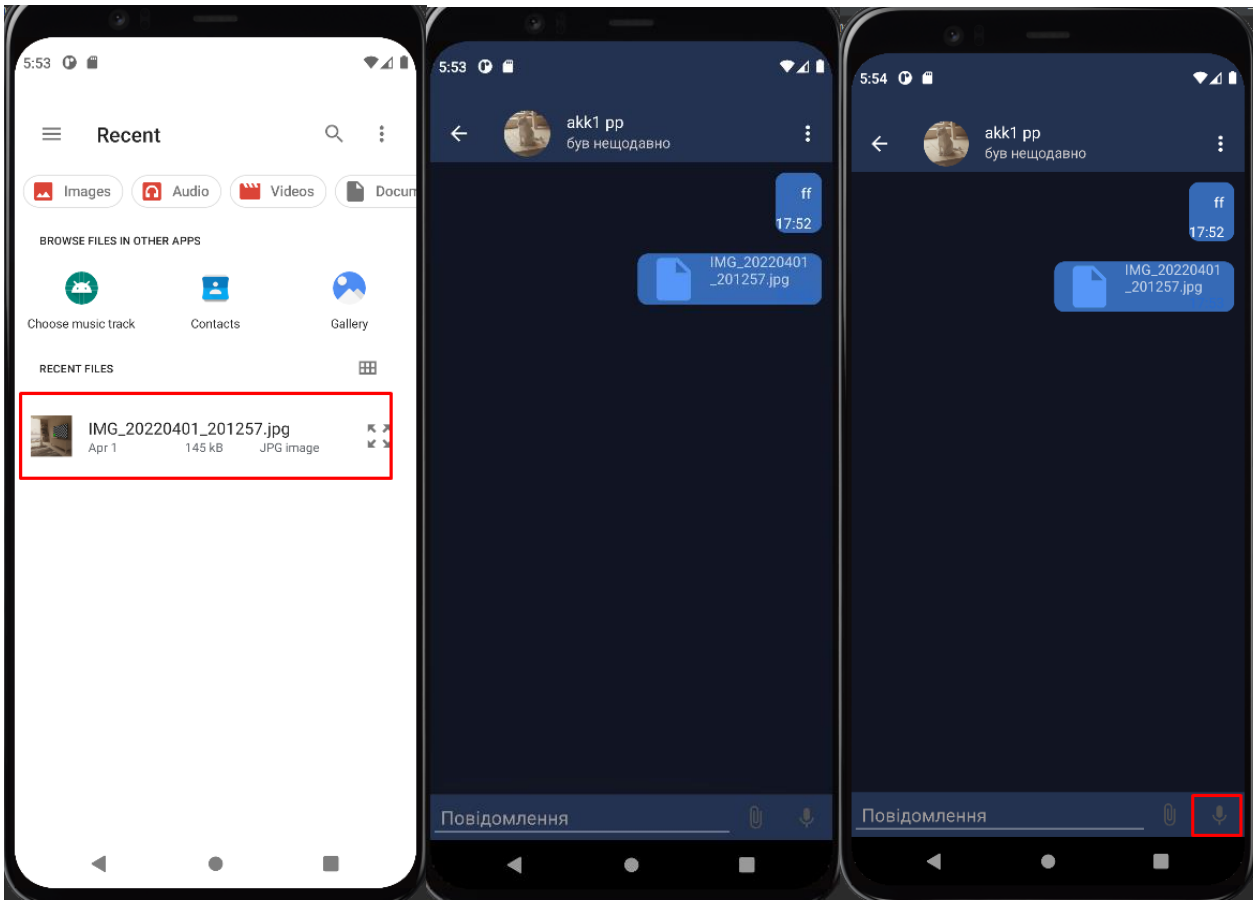

Рисунок 3.37 – Чати 2/3

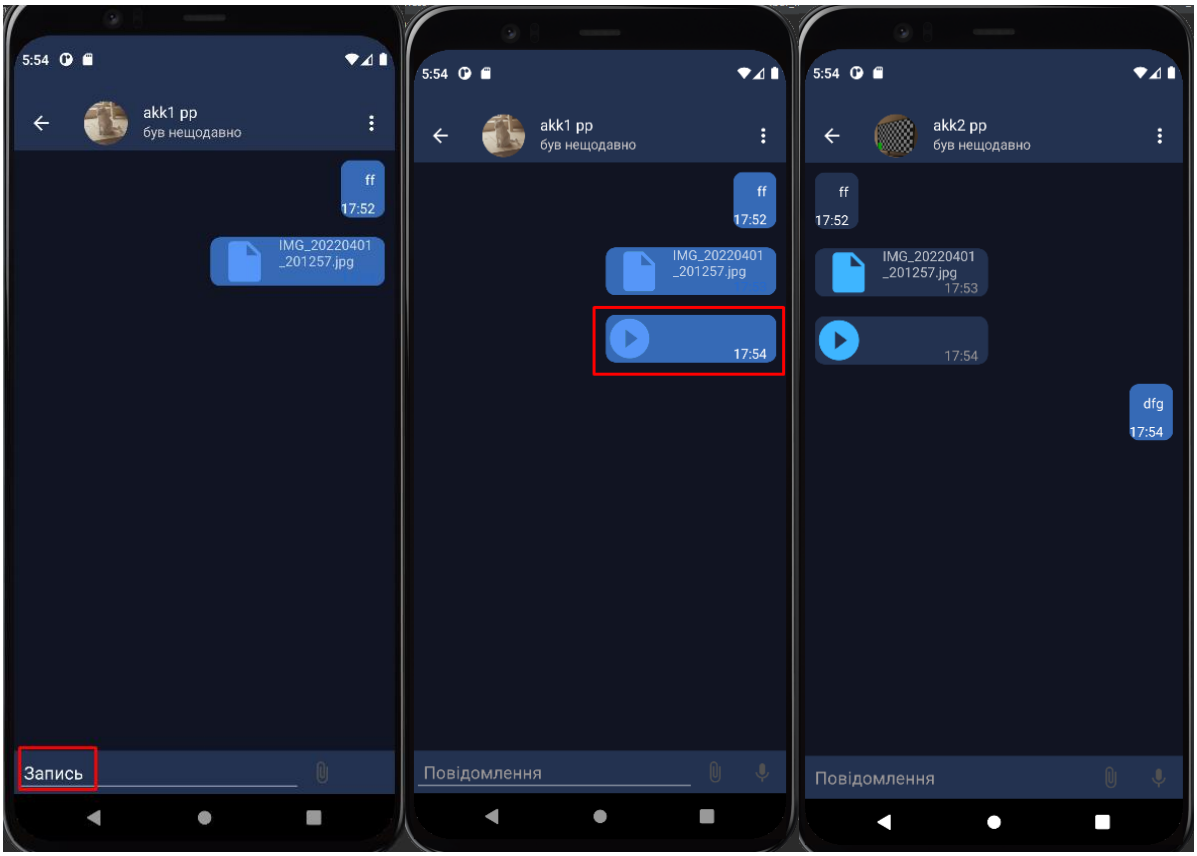

Рисунок 3.38 – Чати 3/3

Вся інформація зберігається в базі даних Firebase (рис. 3.39-3.40).

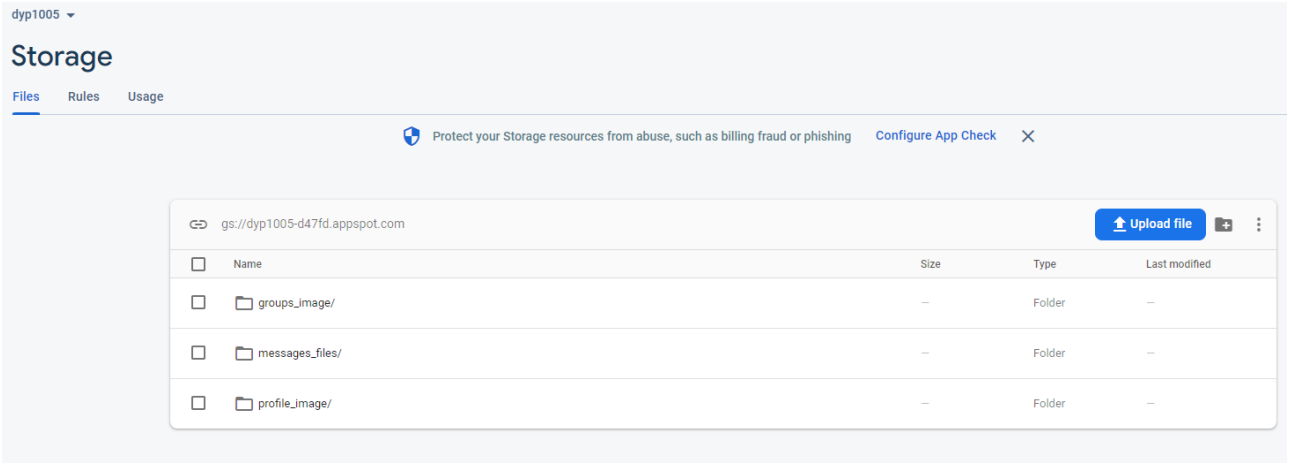

Рисунок 3.39 – Останні чати

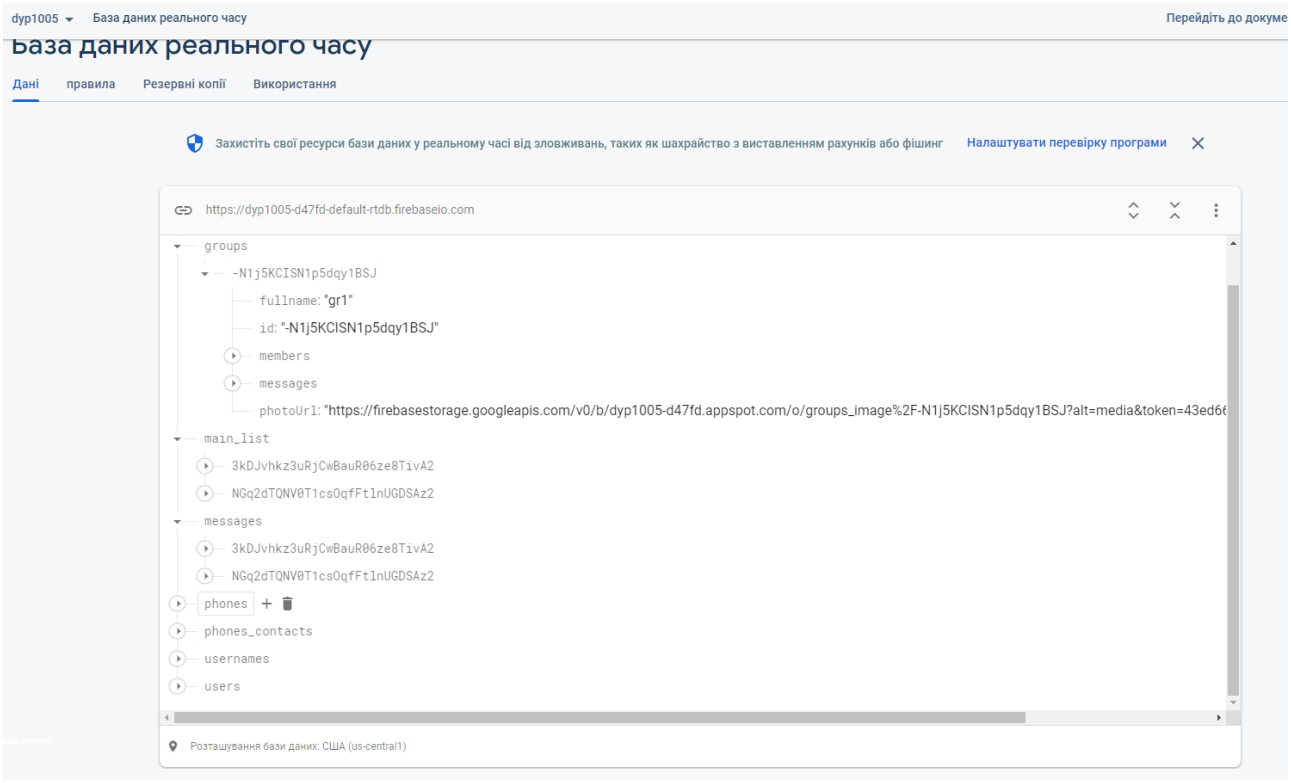

Рисунок 3.40 – Останні чати

3.4 Тестування

Для перевірки працездатності застосунку використовувались віртуальні девайси з ОС Android. В Android Studio можна легко і швидко вибрати будь-який пристрій з списку доступних з різними версіями Android. В основному тестування працездатності проводилось на віртуальному Pixel 5 API 30 (рис. 3.41) версія Android 11.0. Після того, як було отримано працюючий застосунок, його було протестовано на віртуальному Nexus 5 API 30 (рис. 3.42) і на віртуальному Pixel 3a API 30x86 (рис. 3.43). Роботу застосунку відразу на трьох пристоях можна побачити на рис. 3.44 і рис. 3.45.

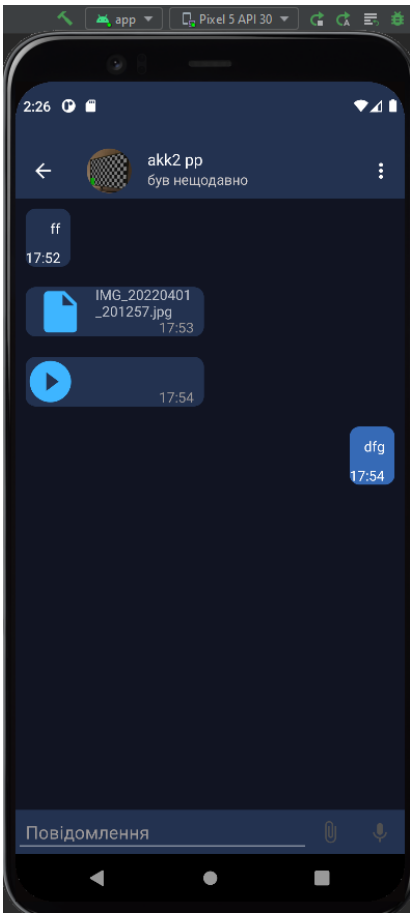

Рисунок 3.41 – Тестування на віртуальному Pixel 5 API 30

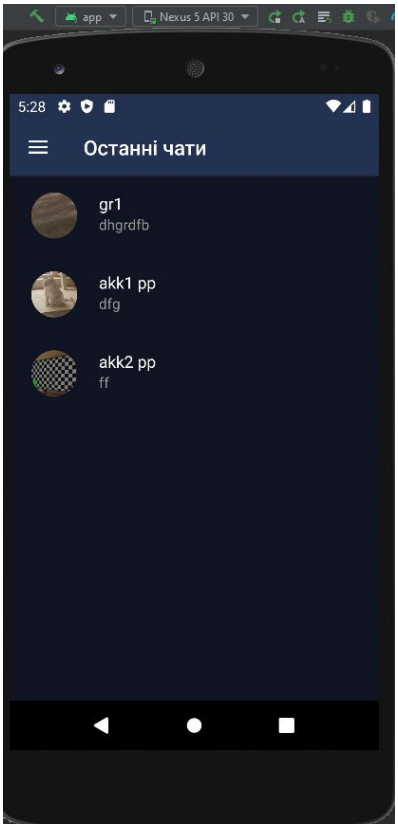

Рисунок 3.42 – Тестування на віртуальному Nexus 5 API 30

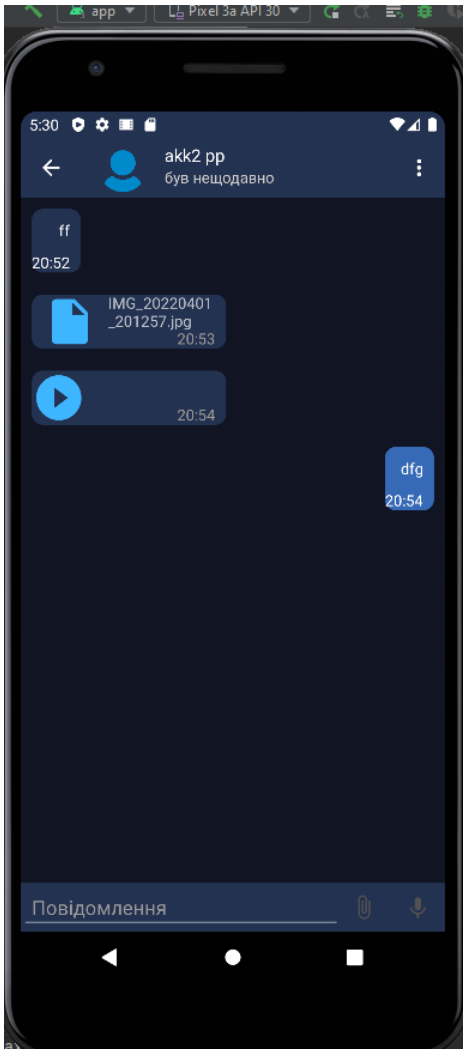

Рисунок 3.43 – Тестування на віртуальному Pixel 3a API 30x86

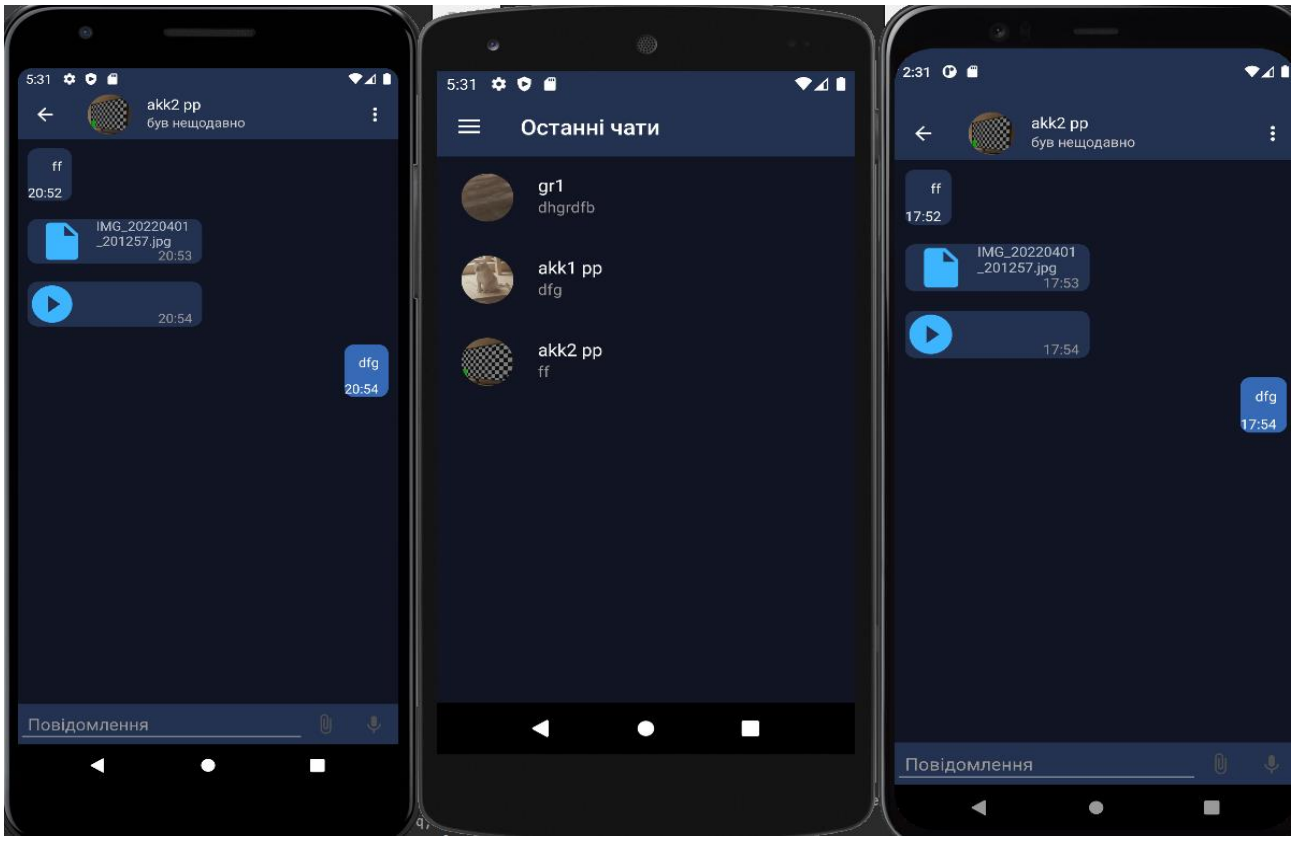

Рисунок 3.44 – Робота застосунку відразу на трьох віртуальних девайсах Pixel 5 API 30, Nexus 5 API 30 і Pixel 3a API 30x86

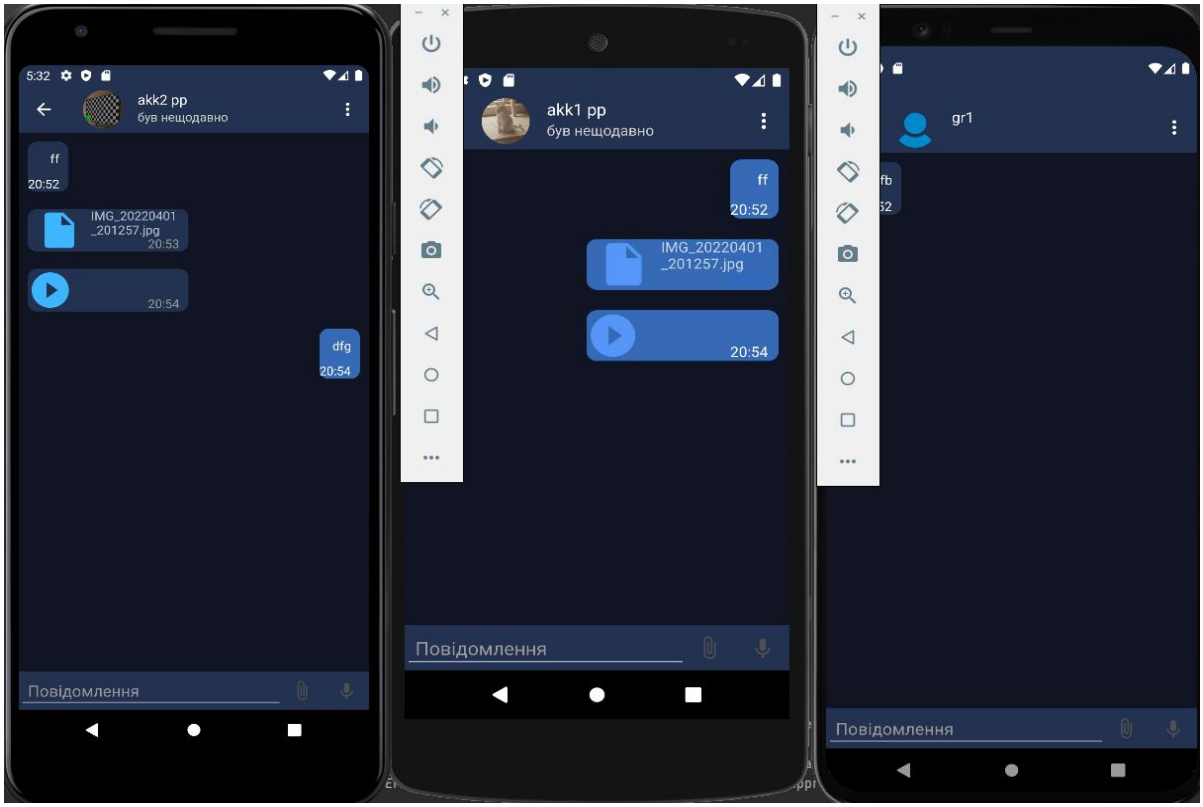

Рисунок 3.45 – Робота застосунку відразу на трьох віртуальних девайсах Pixel 5 API 30, Nexus 5 API 30 і Pixel 3a API 30x86

Після всіх попередніх тестувань застосунок було завантажено на власний смартон Realme 6 з версією Android 11.0 (рис. 3.46). Більш детально працездатність застосунку на реальному пристрої описано в розділі 3.3.

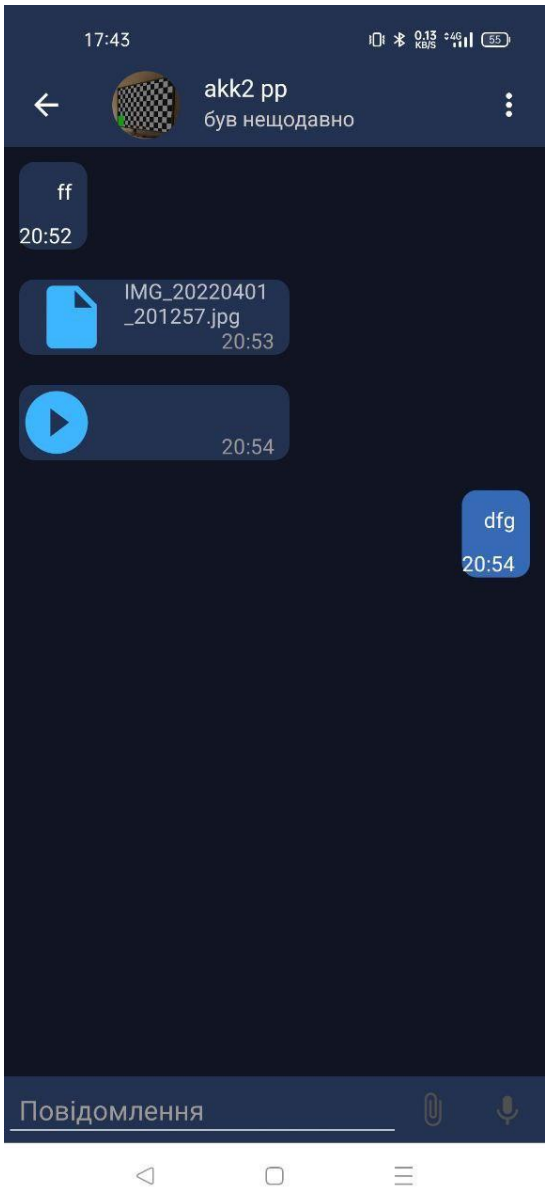

Рисунок 3.46 – Робота застосунку на Realme 6

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було досягнуто мету та отримано такі результати:

1. Було проведено аналіз платформ для мобільних пристроїв, обрано найбільш популярну, доведено актуальність створення даного застосунку і розібрано як додати застосунок до Play Market.

2. Було створено застосунок, що реалізує такі функції:

- додавання контактів, які користуються застосунком, до списку контактів;
- приватні чати;
- групові чати;
- можливість відправлення файлів, картинок, голосових повідомлень.

3. Застосунок було протестовано і доведено його працездатність як на стандартних емуляторах, взятих з SDK Android, так і на реальних пристроях на платформі Android.

Можна виокремити такі напрямки подальшого розвитку застосунку:

1. Додати застосунку доступ до геолокації і створення міток.

2. Додати можливість пошуку публічних каналів.

4. Додати можливість закріплювати обрані чати.

5. Реалізувати можливість користуватися декількома аккаунтами з одного пристрою без виходу з аккаунта

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android (operating system). URL: https://en.wikipedia.org/wiki/Android_(operating_system) (дата звернення: 15.02.2022).

2. Statcounter Globalstat. URL: https://gs.statcounter.com/ (дата звернення: 18.02.2022).

3. Руководство по языку Kotlin. URL: https://metanit.com/kotlin/tutorial/" (дата звернення: 14.02.2022).

4. Работа с XML. URL: "https://metanit.com/java/android/23.1.php" (дата звернення: 20.02.2022).

5. Альтернатива Android: топ-8 мобильных операционных систем. URL: https://okdk.ru/alternativa-android-top-8-mobilnyh-operacionnyh-sistem-v-2019-godu/" (дата звернення: 20.03.2022).

6. Розробка мобільних додатків від А до Я: повний гайд. URL: "https://dan-it.com.ua/uk/rozrobka-mobilnih-dodatkiv-vid-a-do-ja-povnij-gajd/" (дата звернення: 10.02.2022).

7. Аделекан И. Kotlin: Программирование на примерах. Санкт-Петербург: БХВ-Петербург, 2020. 432 с.

8. Документація firebase. URL: https://firebase.google.com/docs" (дата звернення: 1.02.2022).

9. Документація по роботі з зображеннями firebase. URL: https://habr.com/ru/post/452318/" (дата звернення: 2.02.2022).

10. Документація по Navigation draver. URL: https://material.io/components/navigation-drawer#usage" (дата звернення: 4.02.2022).

11. Документація по CircleImageView. URL: https://github.com/hdodenhof/CircleImageView " (дата звернення: 5.02.2022).

12. Документація по Android Image Cropper. URL: https://github.com/ArthurHub/Android-Image-Cropper" (дата звернення: 6.02.2022).

13. Документація по picasso. URL: http://developer.alexanderklimov.ru/android/library/picasso.php" (дата звернення: 7.02.2022).
14. Документація по корунтинам в Kotlin. URL: https://metanit.com/kotlin/tutorial/8.1.php" (дата звернення: 2.02.2022).
15. Теорія по корунтинам в Kotlin. URL: https://blog.skillfactory.ru/glossary/coroutine/" (дата звернення: 3.02.2022).

ДОДАТОК А

```
package com.example.dyp.db

import android.net.Uri
import com.example.dyp.R
import com.example.dyp.models.GeneralModel
import com.example.dyp.models.UserModel
import com.example.dyp.utilits.APP_ACTIVITY
import com.example.dyp.utilits.ValueEventListener
import com.example.dyp.utilits.TYPE_GROUP
import com.example.dyp.utilits.showToast
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.FirebaseDatabase
import com.google.firebase.database.ServerValue
import com.google.firebase.storage.FirebaseStorage
import com.google.firebase.storage.StorageReference
import java.io.File
import java.util.ArrayList
import java.util.HashMap

fun initializationFirebase() {
    AUTH =
        FirebaseAuth.getInstance()
    REF_DATABASE_ROOT = FirebaseDatabase.getInstance().reference
    USER =
        UserModel()
    CURRENT_UID = AUTH.currentUser?.uid.toString()
```

```
    REF_STORAGE_ROOT = FirebaseStorage.getInstance().reference
}
inline fun loadUrltoDB(url: String, crossinline function: () -> Unit) {
    REF_DATABASE_ROOT.child(USERS).child(
        CURRENT_UID
    )
        .child(CHILD_PHOTO_URL).setValue(url)
        .addOnSuccessListener { function() }
        .addOnFailureListener { showToast(it.message.toString()) }
}

inline fun getUrlFromDB(path: StorageReference, crossinline function: (url: String) -
> Unit) {
    path.downloadUrl
        .addOnSuccessListener { function(it.toString()) }
        .addOnFailureListener { showToast(it.message.toString()) }
}

inline fun loadfiletoDB(uri: Uri, path: StorageReference, crossinline function: () ->
Unit) {
    path.putFile(uri)
        .addOnSuccessListener { function() }
        .addOnFailureListener { showToast(it.message.toString()) }

}

inline fun initUser(crossinline function: () -> Unit) {
    REF_DATABASE_ROOT.child(USERS).child(
        CURRENT_UID
```

```
    )
        .addListenerForSingleValueEvent(ValueEventListener {
            USER =
                it.getValue(UserModel::class.java)
                    ?: UserModel()
            if (USER.username.isEmpty()) {
                USER.username =
                    CURRENT_UID
            }
            function()
        })
}

fun updatePhonesToDB(arrayContacts: ArrayList<GeneralModel>) {
    if (AUTH.currentUser != null) {

REF_DATABASE_ROOT.child(NODE_PHONES).addListenerForSingleValueEven
t(
            ValueEventListener {
                it.children.forEach { snapshot ->
                    arrayContacts.forEach { contact ->
                        if (snapshot.key == contact.phone) {
                            REF_DATABASE_ROOT.child(
                                NODE_PHONES_CONTACTS
                            ).child(CURRENT_UID)
                                .child(snapshot.value.toString())
                                .child(CHILD_ID)
                                .setValue(snapshot.value.toString())
                                .addOnFailureListener {
```

```
                        showToast(
                            it.message.toString()
                        )
                    }

                REF_DATABASE_ROOT.child(
                    NODE_PHONES_CONTACTS
                ).child(CURRENT_UID)
                    .child(snapshot.value.toString())
                    .child(CHILD_FULLNAME)
                    .setValue(contact.fullname)
                    .addOnFailureListener {
                        showToast(
                            it.message.toString()
                        )
                    }
            }
        }
    }
}
```

```
fun sendMessage(message: String, receivingUserID: String, typeText: String,
function: () -> Unit) {
    val refDialogUser = "$NODE_MESSAGES/$CURRENT_UID/$receivingUserID"
```

```
    val refDialogReceivingUser =
"$NODE_MESSAGES/$receivingUserID/$CURRENT_UID"
    val messageKey = REF_DATABASE_ROOT.child(refDialogUser).push().key

    val mapMessage = hashMapOf<String, Any>()
    mapMessage[CHILD_FROM] =
        CURRENT_UID
    mapMessage[CHILD_TYPE] = typeText
    mapMessage[CHILD_TEXT] = message
    mapMessage[CHILD_ID] = messageKey.toString()
    mapMessage[CHILD_TIMESTAMP] =
        ServerValue.TIMESTAMP

    val mapDialog = hashMapOf<String, Any>()
    mapDialog["$refDialogUser/$messageKey"] = mapMessage
    mapDialog["$refDialogReceivingUser/$messageKey"] = mapMessage

    REF_DATABASE_ROOT
        .updateChildren(mapDialog)
        .addOnSuccessListener { function() }
        .addOnFailureListener { showToast(it.message.toString()) }

}
fun DataSnapshot.getCommonModel(): GeneralModel =
    this.getValue(GeneralModel::class.java) ?: GeneralModel()

fun DataSnapshot.getUserModel(): UserModel =
    this.getValue(UserModel::class.java) ?: UserModel()
```

```
private fun deleteOldUsername(newUserName: String) {
    REF_DATABASE_ROOT.child(NODE_USERNAMES).child(
        USER.username
    ).removeValue()
        .addOnSuccessListener {
            showToast(
                APP_ACTIVITY.getString(
                    R.string.toast_data_update
                )
            )
            APP_ACTIVITY.supportFragmentManager.popBackStack()
            USER.username = newUserName
        }.addOnFailureListener { showToast(it.message.toString()) }
}

fun setBioToDatabase(newBio: String) {
    REF_DATABASE_ROOT.child(USERS).child(
        CURRENT_UID
    ).child(CHILD_INFO)
        .setValue(newBio)
        .addOnSuccessListener {
            showToast(
                APP_ACTIVITY.getString(
                    R.string.toast_data_update
                )
            )
            USER.bio = newBio
```

```
            APP_ACTIVITY.supportFragmentManager.popBackStack()
        }.addOnFailureListener { showToast(it.message.toString()) }
}
fun updateCurrentUsername(newUserName: String) {
    REF_DATABASE_ROOT.child(USERS).child(
        CURRENT_UID
    ).child(CHILD_USERNAME)
        .setValue(newUserName)
        .addOnCompleteListener {
            if (it.isSuccessful) {
                showToast(
                    APP_ACTIVITY.getString(
                        R.string.toast_data_update
                    )
                )
                deleteOldUsername(newUserName)
            } else {
                showToast(it.exception?.message.toString())
            }
        }
}


fun sendMessageAsFile(
    receivingUserID: String,
    fileUrl: String,
    messageKey: String,
    typeMessage: String,
```

```
    filename: String
) {

    val refDialogUser = "$NODE_MESSAGES/$CURRENT_UID/$receivingUserID"
    val refDialogReceivingUser =
"$NODE_MESSAGES/$receivingUserID/$CURRENT_UID"

    val mapMessage = hashMapOf<String, Any>()
    mapMessage[CHILD_FROM] = CURRENT_UID
    mapMessage[CHILD_TYPE] = typeMessage
    mapMessage[CHILD_ID] = messageKey
    mapMessage[CHILD_TIMESTAMP] = ServerValue.TIMESTAMP
    mapMessage[CHILD_FILE_URL] = fileUrl
    mapMessage[CHILD_TEXT] = filename

    val mapDialog = hashMapOf<String, Any>()
    mapDialog["$refDialogUser/$messageKey"] = mapMessage
    mapDialog["$refDialogReceivingUser/$messageKey"] = mapMessage

    REF_DATABASE_ROOT
        .updateChildren(mapDialog)
        .addOnFailureListener { showToast(it.message.toString()) }
}



fun setNameToDatabase(fullname: String) {
    REF_DATABASE_ROOT.child(
        USERS
    ).child(CURRENT_UID).child(
```

```
        CHILD_FULLNAME
    ).setValue(fullname)
        .addOnSuccessListener {
            showToast(
                APP_ACTIVITY.getString(
                    R.string.toast_data_update
                )
            )
            USER.fullname = fullname
            APP_ACTIVITY.mAppDrawer.updHeader()
            APP_ACTIVITY.supportFragmentManager.popBackStack()
        }.addOnFailureListener { showToast(it.message.toString()) }
}
fun uploadFiletoStorage(
    uri: Uri,
    messageKey: String,
    receivedID: String,
    typeMessage: String,
    filename: String = ""
) {
    val path = REF_STORAGE_ROOT.child(
        FILE_FOLDER
    ).child(messageKey)
    loadfiletoDB(uri, path) {
        getUrlFromDB(path) {
            sendMessageAsFile(
                receivedID,
                it,
```

```
                messageKey,
                typeMessage,
                filename
            )
        }
    }
}
fun getMessageKey(id: String) = REF_DATABASE_ROOT.child(
    NODE_MESSAGES
).child(CURRENT_UID)
    .child(id).push().key.toString()
fun downloadFilewithDB(mFile: File, fileUrl: String, function: () -> Unit) {
    val path = REF_STORAGE_ROOT.storage.getReferenceFromUrl(fileUrl)
    path.getFile(mFile)
        .addOnSuccessListener { function() }
        .addOnFailureListener { showToast(it.message.toString()) }
}

fun saveToMainList(id: String, type: String) {
    val referuser = "$NODE_MAIN_LIST/$CURRENT_UID/$id"
    val referrecieved = "$NODE_MAIN_LIST/$id/$CURRENT_UID"

    val mapUser = hashMapOf<String, Any>()
    val mapReceived = hashMapOf<String, Any>()

    mapUser[CHILD_ID] = id
    mapUser[CHILD_TYPE] = type
```

```
    mapReceived[CHILD_ID] = CURRENT_UID
    mapReceived[CHILD_TYPE] = type

    val commonMap = hashMapOf<String, Any>()
    commonMap[referuser] = mapUser
    commonMap[referrecieved] = mapReceived

    REF_DATABASE_ROOT.updateChildren(commonMap)
        .addOnFailureListener { showToast(it.message.toString()) }

}



fun clearChat(id: String, function: () -> Unit) {

REF_DATABASE_ROOT.child(NODE_MESSAGES).child(CURRENT_UID).child
(id)
        .removeValue()
        .addOnFailureListener { showToast(it.message.toString()) }
        .addOnSuccessListener {
            REF_DATABASE_ROOT.child(NODE_MESSAGES).child(id)
                .child(CURRENT_UID)
                .removeValue()
                .addOnSuccessListener { function() }
        }
        .addOnFailureListener { showToast(it.message.toString()) }
}

fun deleteChat(id: String, function: () -> Unit) {
```

```
REF_DATABASE_ROOT.child(NODE_MAIN_LIST).child(CURRENT_UID).chil
d(id).removeValue()
        .addOnFailureListener { showToast(it.message.toString()) }
        .addOnSuccessListener { function() }
}

fun createGroupinDB(
    nameGroup: String,
    uri: Uri,
    listContacts: List<GeneralModel>,
    function: () -> Unit
) {

    val keyGroup =
REF_DATABASE_ROOT.child(NODE_GROUPS).push().key.toString()
    val path = REF_DATABASE_ROOT.child(NODE_GROUPS).child(keyGroup)
    val pathStorage =
REF_STORAGE_ROOT.child(FOLDER_GROUPS_IMAGE).child(keyGroup)

    val mapData = hashMapOf<String, Any>()
    mapData[CHILD_ID] = keyGroup
    mapData[CHILD_FULLNAME] = nameGroup
    mapData[CHILD_PHOTO_URL] = "empty"
    val mapMembers = hashMapOf<String, Any>()
    listContacts.forEach {
        mapMembers[it.id] = USER_MEMBER
    }
    mapMembers[CURRENT_UID] = USER_CREATOR
```

```
    mapData[NODE_MEMBERS] = mapMembers

    path.updateChildren(mapData)
        .addOnSuccessListener {

            if (uri != Uri.EMPTY) {
                loadfiletoDB(uri, pathStorage) {
                    getUrlFromDB(pathStorage) {
                        path.child(CHILD_PHOTO_URL).setValue(it)
                        MainlistGrouplistener(mapData, listContacts) {
                            function()
                        }
                    }
                }
            } else {
                MainlistGrouplistener(mapData, listContacts) {
                    function()
                }
            }

        }
        .addOnFailureListener { showToast(it.message.toString()) }


}
```

```
fun sendGroupMessage(message: String, groupID: String, typeText: String, function:
() -> Unit) {

    var refermessage = "$NODE_GROUPS/$groupID/$NODE_MESSAGES"
    val refmessagekey = REF_DATABASE_ROOT.child(refermessage).push().key

    val mapMessageKey = hashMapOf<String, Any>()
    mapMessageKey[CHILD_FROM] =
        CURRENT_UID
    mapMessageKey[CHILD_TYPE] = typeText
    mapMessageKey[CHILD_TEXT] = message
    mapMessageKey[CHILD_ID] = refmessagekey.toString()
    mapMessageKey[CHILD_TIMESTAMP] =
        ServerValue.TIMESTAMP

    REF_DATABASE_ROOT.child(refermessage).child(refmessagekey.toString())
        .updateChildren(mapMessageKey)
        .addOnSuccessListener { function() }
        .addOnFailureListener { showToast(it.message.toString()) }

}
fun MainlistGrouplistener(
    mapData: HashMap<String, Any>,
    listContacts: List<GeneralModel>,
    function: () -> Unit
) {
    val path = REF_DATABASE_ROOT.child(NODE_MAIN_LIST)
    val map = hashMapOf<String, Any>()
```

```
    map[CHILD_ID] = mapData[CHILD_ID].toString()
    map[CHILD_TYPE] = TYPE_GROUP
    listContacts.forEach {
        path.child(it.id).child(map[CHILD_ID].toString()).updateChildren(map)
    }

path.child(CURRENT_UID).child(map[CHILD_ID].toString()).updateChildren(map
)
        .addOnSuccessListener { function() }
        .addOnFailureListener { showToast(it.message.toString()) }

}package com.example.dyp.db

import android.view.Menu
import android.view.MenuInflater
import android.view.MenuItem
import androidx.fragment.app.Fragment
import com.example.dyp.MainActivity
import com.example.dyp.R
import com.example.dyp.utilits.Keyboard



open class BaseChangeFragment (layout:Int): Fragment(layout) {


    override fun onStart() {
        super.onStart()
        setHasOptionsMenu(true)
```

```
        (activity as MainActivity).mAppDrawer.disableDrawer()
        Keyboard()
    }

    override fun onStop() {
        super.onStop()
    }

    override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {

        (activity as MainActivity).menuInflater.inflate(R.menu.settings_menu_confirm,
menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {

        when (item.itemId) {
            R.id.settings_confirm_change -> change()
        }
        return true
    }

    open fun change() {

    }
}
package com.example.dyp.fragments.base

import androidx.fragment.app.Fragment
```

```
import com.example.dyp.utilits.APP_ACTIVITY


open class BaseFragment( layout:Int) : Fragment(layout) {

    override fun onStart() {
        super.onStart()
        APP_ACTIVITY.mAppDrawer.disableDrawer()
    }
}
package com.example.dyp.fragments.contacts

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.models.GeneralModel
import com.example.dyp.fragments.base.BaseFragment
import com.example.dyp.fragments.single_chat.SingleChatFragment
import com.example.dyp.utilits.*
import com.firebase.ui.database.FirebaseRecyclerAdapter
import com.firebase.ui.database.FirebaseRecyclerOptions
import com.google.firebase.database.DatabaseReference
import de.hdodenhof.circleimageview.CircleImageView
import kotlinx.android.synthetic.main.contact_item.view.*
```

```
import kotlinx.android.synthetic.main.fragment_contacts.*


class ContactsFragment : BaseFragment(R.layout.fragment_contacts) {

    private lateinit var mRecyclerView: RecyclerView
    private lateinit var mAdapter: FirebaseRecyclerAdapter<GeneralModel,
ContactsHolder>
    private lateinit var mRefContacts: DatabaseReference
    private lateinit var mRefUsers: DatabaseReference
    private lateinit var mRefUsersListener:ValueEventListener
    private  var mapListeners =
hashMapOf<DatabaseReference,ValueEventListener>()

    override fun onResume() {
        super.onResume()
        APP_ACTIVITY.title = "Контакти"
        initRecycleView()
    }

    private fun initRecycleView() {
        mRecyclerView = contacts_recycle_view
        mRefContacts = REF_DATABASE_ROOT.child(
            NODE_PHONES_CONTACTS
        ).child(CURRENT_UID)


        val options = FirebaseRecyclerOptions.Builder<GeneralModel>()
            .setQuery(mRefContacts, GeneralModel::class.java)
```

```
        .build()


    mAdapter = object : FirebaseRecyclerAdapter<GeneralModel,
ContactsHolder>(options) {

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ContactsHolder {

            val view = LayoutInflater.from(parent.context)
                .inflate(R.layout.contact_item, parent, false)
            return ContactsHolder(
                view
            )
        }


        override fun onBindViewHolder(
            holder: ContactsHolder,
            position: Int,
            model: GeneralModel
        ) {
            mRefUsers = REF_DATABASE_ROOT.child(
                USERS
            ).child(model.id)

            mRefUsersListener = ValueEventListener {
                val contact = it.getCommonModel()
```

```
            if (contact.fullname.isEmpty()){
                holder.name.text = model.fullname
            } else holder.name.text = contact.fullname

            holder.status.text = contact.state
            holder.photo.downloadAndSetImage(contact.photoUrl)
            holder.itemView.setOnClickListener { replaceFragment(
                SingleChatFragment(
                    model
                )
            ) }
        }

        mRefUsers.addValueEventListener(mRefUsersListener)
        mapListeners[mRefUsers] = mRefUsersListener
    }
}

mRecyclerView.adapter = mAdapter
mAdapter.startListening()

}


class ContactsHolder(view: View) : RecyclerView.ViewHolder(view) {
    val name: TextView = view.contact_fullname1
    val status: TextView = view.contact_status
    val photo: CircleImageView = view.contact_photo
```

```
    }

    override fun onPause() {
        super.onPause()
        mAdapter.stopListening()
        println()
        mapListeners.forEach {
            it.key.removeEventListener(it.value)
        }
        println()
    }
}
package com.example.dyp.fragments.main_list

import androidx.fragment.app.Fragment
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.models.GeneralModel
import com.example.dyp.utilits.*
import kotlinx.android.synthetic.main.fragment_main_list.*

class MainListFragment : Fragment(R.layout.fragment_main_list) {

    private lateinit var mRecyclerView: RecyclerView
    private lateinit var mAdapter: MainListAdapter
    private val mRefMainList =
REF_DATABASE_ROOT.child(NODE_MAIN_LIST).child(CURRENT_UID)
    private val mRefUsers = REF_DATABASE_ROOT.child(USERS)
```

```
    private val mRefMessages =
REF_DATABASE_ROOT.child(NODE_MESSAGES).child(CURRENT_UID)
    private var mListItems = listOf<GeneralModel>()

    override fun onResume() {
        super.onResume()
        APP_ACTIVITY.title = "Останні чати"
        APP_ACTIVITY.mAppDrawer.enableDrawer()
        Keyboard()
        initRecyclerView()
    }

    private fun initRecyclerView() {
        mRecyclerView = main_list_recycle_view
        mAdapter = MainListAdapter()



        mRefMainList.addListenerForSingleValueEvent(ValueEventListener {
dataSnapshot ->
            mListItems = dataSnapshot.children.map { it.getCommonModel() }
            mListItems.forEach { model ->

                when(model.type){
                    TYPE_CHAT -> showChat(model)
                    TYPE_GROUP -> showGroup(model)
                }

            }
        })
```

```
        mRecyclerView.adapter = mAdapter
    }

    private fun showGroup(model: GeneralModel) {

        REF_DATABASE_ROOT.child(NODE_GROUPS).child(model.id)
            .addListenerForSingleValueEvent(ValueEventListener { dataSnapshot1 ->
                val newModel = dataSnapshot1.getCommonModel()


REF_DATABASE_ROOT.child(NODE_GROUPS).child(model.id).child(NODE_MESSAGES)
                    .limitToLast(1)
                    .addListenerForSingleValueEvent(ValueEventListener { dataSnapshot2 ->
                        val tempList = dataSnapshot2.children.map { it.getCommonModel() }

                        if (tempList.isEmpty()){
                            newModel.lastMessage = "Чат очищено"
                        } else {
                            newModel.lastMessage = tempList[0].text
                        }
                        newModel.type = TYPE_GROUP
                        mAdapter.updateListItems(newModel)
                    })
            })

    }
```

```
    private fun showChat(model: GeneralModel) {

        mRefUsers.child(model.id)
            .addListenerForSingleValueEvent(ValueEventListener { dataSnapshot1 ->
                val newModel = dataSnapshot1.getCommonModel()


                mRefMessages.child(model.id).limitToLast(1)
                    .addListenerForSingleValueEvent(ValueEventListener { dataSnapshot2 -
>
                        val tempList = dataSnapshot2.children.map { it.getCommonModel() }

                        if (tempList.isEmpty()){
                            newModel.lastMessage = "Чат очищено"
                        } else {
                            newModel.lastMessage = tempList[0].text
                        }


                        if (newModel.fullname.isEmpty()) {
                            newModel.fullname = newModel.phone
                        }

                        newModel.type = TYPE_CHAT
                        mAdapter.updateListItems(newModel)
                    })
            })
    }
```

```
}
package com.example.dyp.fragments.register

import androidx.fragment.app.Fragment
import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.utilits.*
import com.google.firebase.auth.PhoneAuthProvider
import kotlinx.android.synthetic.main.fragment_enter_code.*



class EnterCodeFragment(val phoneNumber: String, val id: String) :
    Fragment(R.layout.fragment_enter_code) {


    override fun onStart() {
        super.onStart()
        APP_ACTIVITY.title = phoneNumber
        register_input_code.addTextChangedListener(TextWatcher {
            val string = register_input_code.text.toString()
            if (string.length == 6) {
                enterCode()
            }
        })
    }

    private fun enterCode() {
```

```
val code = register_input_code.text.toString()
val credential = PhoneAuthProvider.getCredential(id, code)
AUTH.signInWithCredential(credential).addOnCompleteListener { task ->
    if (task.isSuccessful) {
        val uid = AUTH.currentUser?.uid.toString()
        val dateMap = mutableMapOf<String, Any>()
        dateMap[CHILD_ID] = uid
        dateMap[CHILD_PHONE] = phoneNumber


        REF_DATABASE_ROOT.child(USERS).child(uid)
            .addListenerForSingleValueEvent(ValueEventListener{

                if (!it.hasChild(CHILD_USERNAME)){
                    dateMap[CHILD_USERNAME] = uid
                }

                REF_DATABASE_ROOT.child(
                    NODE_PHONES
                ).child(phoneNumber).setValue(uid)
                    .addOnFailureListener { showToast(it.message.toString()) }
                    .addOnSuccessListener {
                        REF_DATABASE_ROOT.child(
                            USERS
                        ).child(uid).updateChildren(dateMap)
                            .addOnSuccessListener {
                                showToast("Ласкаво просимо")
```

```
                        restartActivity()
                    }
                    .addOnFailureListener { showToast(it.message.toString()) }
                }
            })



        } else showToast(task.exception?.message.toString())
    }
  }
}

package com.example.dyp.fragments.groups

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
import com.example.dyp.models.GeneralModel
import com.example.dyp.utilits.downloadAndSetImage
import de.hdodenhof.circleimageview.CircleImageView
import kotlinx.android.synthetic.main.add_contacts_item.view.*

class AddContactsAdapter :
RecyclerView.Adapter<AddContactsAdapter.AddContactsHolder>() {
```

```
    private var listItems = mutableListOf<GeneralModel>()

    class AddContactsHolder(view: View) : RecyclerView.ViewHolder(view) {
        val itemName: TextView = view.add_contacts_item_name
        val itemLastMessage: TextView = view.add_contacts_last_message
        val itemPhoto: CircleImageView = view.add_contacts_item_photo
        val itemChoice:CircleImageView = view.add_contacts_item_choice
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
AddContactsHolder {
        val view =
            LayoutInflater.from(parent.context).inflate(R.layout.add_contacts_item,
parent, false)

        val holder = AddContactsHolder(view)
        holder.itemView.setOnClickListener {
          if (listItems[holder.adapterPosition].choice){
              holder.itemChoice.visibility = View.INVISIBLE
              listItems[holder.adapterPosition].choice = false

AddContactsFragment.listContacts.remove(listItems[holder.adapterPosition])
          } else {
              holder.itemChoice.visibility = View.VISIBLE
              listItems[holder.adapterPosition].choice = true
              AddContactsFragment.listContacts.add(listItems[holder.adapterPosition])
          }
        }
        return holder
```

```
    }

    override fun getItemCount(): Int = listItems.size

    override fun onBindViewHolder(holder: AddContactsHolder, position: Int) {
        holder.itemName.text = listItems[position].fullname
        holder.itemLastMessage.text = listItems[position].lastMessage
        holder.itemPhoto.downloadAndSetImage(listItems[position].photoUrl)
    }

    fun updateListItems(item:GeneralModel){
        listItems.add(item)
        notifyItemInserted(listItems.size)
    }
}
package com.example.dyp.fragments.groups

import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.models.GeneralModel
import com.example.dyp.fragments.base.BaseFragment
import com.example.dyp.utilits.*
import kotlinx.android.synthetic.main.fragment_add_contacts.*



class AddContactsFragment : BaseFragment(R.layout.fragment_add_contacts) {
```

```
    private lateinit var mRecyclerView: RecyclerView
    private lateinit var mAdapter: AddContactsAdapter
    private val mRefContactsList =
REF_DATABASE_ROOT.child(NODE_PHONES_CONTACTS).child(CURRENT
_UID)
    private val mRefUsers = REF_DATABASE_ROOT.child(USERS)
    private val mRefMessages =
REF_DATABASE_ROOT.child(NODE_MESSAGES).child(CURRENT_UID)
    private var mListItems = listOf<GeneralModel>()

    override fun onResume() {
        listContacts.clear()
        super.onResume()
        APP_ACTIVITY.title = "Додати учасника"
        Keyboard()
        initRecyclerView()
        add_contacts_btn_next.setOnClickListener {
            if (listContacts.isEmpty()) showToast("Додати учасника")
            else replaceFragment(CreateGroupFragment(listContacts))
        }
    }

    private fun initRecyclerView() {
        mRecyclerView = add_contacts_recycle_view
        mAdapter = AddContactsAdapter()


        mRefContactsList.addListenerForSingleValueEvent(ValueEventListener {
dataSnapshot ->
```

```
        mListItems = dataSnapshot.children.map { it.getCommonModel() }
        mListItems.forEach { model ->

            mRefUsers.child(model.id)
                .addListenerForSingleValueEvent(ValueEventListener { dataSnapshot1 -
>
                    val newModel = dataSnapshot1.getCommonModel()



                    mRefMessages.child(model.id).limitToLast(1)
                        .addListenerForSingleValueEvent(ValueEventListener {
dataSnapshot2 ->
                            val tempList = dataSnapshot2.children.map {
it.getCommonModel() }

                            if (tempList.isEmpty()){
                                newModel.lastMessage = "Чат очищено"
                            } else {
                                newModel.lastMessage = tempList[0].text
                            }



                            if (newModel.fullname.isEmpty()) {
                                newModel.fullname = newModel.phone
                            }
                            mAdapter.updateListItems(newModel)
                        })
                })
        }
```

```
        })

        mRecyclerView.adapter = mAdapter
    }

    companion object{
        val listContacts = mutableListOf<GeneralModel>()
    }
}
package com.example.dyp.fragments.groups

import android.app.Activity
import android.content.Intent
import android.net.Uri
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.models.GeneralModel
import com.example.dyp.fragments.base.BaseFragment
import com.example.dyp.fragments.main_list.MainListFragment
import com.example.dyp.utilits.*
import com.theartofdev.edmodo.cropper.CropImage
import com.theartofdev.edmodo.cropper.CropImageView
import kotlinx.android.synthetic.main.fragment_create_group.*

class CreateGroupFragment(private var listContacts:List<GeneralModel>)
    :BaseFragment(R.layout.fragment_create_group) {
```

```
private lateinit var mRecyclerView: RecyclerView
private lateinit var mAdapter: AddContactsAdapter
private var mUri = Uri.EMPTY

override fun onResume() {
    super.onResume()
    APP_ACTIVITY.title = getString(R.string.create_group)
    Keyboard()
    initRecyclerView()
    create_group_photo.setOnClickListener { addPhoto()  }
    create_group_btn_complete.setOnClickListener {
        val nameGroup = create_group_input_name.text.toString()
        if (nameGroup.isEmpty()){
            showToast("Введіте імя")
        } else {
            createGroupinDB(nameGroup,mUri,listContacts){
                replaceFragment(MainListFragment())
            }
        }
    }
    create_group_input_name.requestFocus()
    create_group_counts.text = getPlurals(listContacts.size)
}


private fun addPhoto() {
    CropImage.activity()
        .setAspectRatio(1, 1)
```

```
            .setRequestedSize(250, 250)
            .setCropShape(CropImageView.CropShape.OVAL)
            .start(APP_ACTIVITY,this)
    }

    private fun initRecyclerView() {
        mRecyclerView = create_group_recycle_view
        mAdapter = AddContactsAdapter()
        mRecyclerView.adapter = mAdapter
        listContacts.forEach {  mAdapter.updateListItems(it) }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {

        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE
            && resultCode == Activity.RESULT_OK && data != null
        ) {
             mUri = CropImage.getActivityResult(data).uri
            create_group_photo.setImageURI(mUri)
        }
    }
}
package com.example.dyp.fragments.info

import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
import com.example.dyp.fragments.base.BaseFragment
```

```
import com.example.dyp.utilits.*
import kotlinx.android.synthetic.main.fragment_info.*


class InfoFragment : BaseFragment(R.layout.fragment_info) {

    private lateinit var mRecyclerView: RecyclerView


    override fun onResume() {
        super.onResume()
        APP_ACTIVITY.title = "Правила"
        initRecycleView()
    }

    private fun initRecycleView() {
        mRecyclerView = info_recycle_view

    }

    override fun onPause() {
        super.onPause()

        println()
    }
}

package com.example.dyp.fragments.main_list
```

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
import com.example.dyp.models.GeneralModel
import com.example.dyp.fragments.groups.GroupChatFragment
import com.example.dyp.fragments.single_chat.SingleChatFragment
import com.example.dyp.utilits.TYPE_CHAT
import com.example.dyp.utilits.TYPE_GROUP
import com.example.dyp.utilits.downloadAndSetImage
import com.example.dyp.utilits.replaceFragment
import de.hdodenhof.circleimageview.CircleImageView
import kotlinx.android.synthetic.main.main_list_item.view.*

class MainListAdapter :
RecyclerView.Adapter<MainListAdapter.MainListHolder>() {

    private var listItems = mutableListOf<GeneralModel>()

    class MainListHolder(view: View) : RecyclerView.ViewHolder(view) {
        val itemName: TextView = view.main_list_item_name
        val itemLastMessage: TextView = view.main_list_last_message
        val itemPhoto: CircleImageView = view.main_list_item_photo
    }
```

```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
MainListHolder {
        val view =
            LayoutInflater.from(parent.context).inflate(R.layout.main_list_item, parent,
false)

        val holder = MainListHolder(view)
        holder.itemView.setOnClickListener {
            when(listItems[holder.adapterPosition].type){
                TYPE_CHAT -
>replaceFragment(SingleChatFragment(listItems[holder.adapterPosition]))
                TYPE_GROUP ->
replaceFragment(GroupChatFragment(listItems[holder.adapterPosition]))
            }
        }
        return holder
    }

    override fun getItemCount(): Int = listItems.size

    override fun onBindViewHolder(holder: MainListHolder, position: Int) {
        holder.itemName.text = listItems[position].fullname
        holder.itemLastMessage.text = listItems[position].lastMessage
        holder.itemPhoto.downloadAndSetImage(listItems[position].photoUrl)
    }

    fun updateListItems(item:GeneralModel){
        listItems.add(item)
        notifyItemInserted(listItems.size)
    }
```

```
}
package com.example.dyp.fragments.register

import androidx.fragment.app.Fragment

import com.example.dyp.R
import com.example.dyp.db.AUTH
import com.example.dyp.utilits.*
import com.google.firebase.FirebaseException
import com.google.firebase.auth.PhoneAuthCredential
import com.google.firebase.auth.PhoneAuthProvider
import kotlinx.android.synthetic.main.fragment_enter_phone_number.*
import java.util.concurrent.TimeUnit



class EnterPhoneNumberFragment :
Fragment(R.layout.fragment_enter_phone_number) {

    private lateinit var mPhoneNumber: String
    private lateinit var mCallback:
PhoneAuthProvider.OnVerificationStateChangedCallbacks

    override fun onStart() {
        super.onStart()

        mCallback = object :
PhoneAuthProvider.OnVerificationStateChangedCallbacks() {
            override fun onVerificationCompleted(credential: PhoneAuthCredential) {
```

```
            AUTH.signInWithCredential(credential).addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    showToast("Ласкаво просимо")
                    restartActivity()
                } else showToast(task.exception?.message.toString())
            }
        }

        override fun onVerificationFailed(p0: FirebaseException) {

            showToast(p0.message.toString())
        }

        override fun onCodeSent(id: String, token:
PhoneAuthProvider.ForceResendingToken) {

            replaceFragment(
                EnterCodeFragment(
                    mPhoneNumber,
                    id
                )
            )
        }
    }
    register_btn_next.setOnClickListener { sendCode() }
}

private fun sendCode() {
```

```
        if (register_input_phone_number.text.toString().isEmpty()) {
            showToast(getString(R.string.register_toast_enter_phone))
        } else {
            authUser()
        }
    }

    private fun authUser() {

        mPhoneNumber = register_input_phone_number.text.toString()
        PhoneAuthProvider.getInstance().verifyPhoneNumber(
            mPhoneNumber,
            60,
            TimeUnit.SECONDS,
            APP_ACTIVITY,
            mCallback
        )
    }
}
package com.example.dyp.fragments.settings

import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.fragments.base.BaseChangeFragment
import kotlinx.android.synthetic.main.fragment_cnage_bio.*
```

```
class ChangeBioFragment : BaseChangeFragment(R.layout.fragment_cnage_bio) {

    override fun onResume() {
        super.onResume()
        settings_input_bio.setText(USER.bio)
    }

    override fun change() {
        super.change()
        val newBio = settings_input_bio.text.toString()
        setBioToDatabase(newBio)
    }
}
package com.example.dyp.fragments.settings

import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.fragments.base.BaseChangeFragment
import com.example.dyp.utilits.*
import kotlinx.android.synthetic.main.fragment_change_name.*




class ChangeNameFragment :
BaseChangeFragment(R.layout.fragment_change_name) {

    override fun onResume() {
        super.onResume()
```

```
        initFullnameList()
    }

    private fun initFullnameList() {
        val fullnameList = USER.fullname.split(" ")
        if (fullnameList.size > 1) {
            settings_input_name.setText(fullnameList[0])
            settings_input_surname.setText(fullnameList[1])
        } else settings_input_name.setText(fullnameList[0])
    }

    override fun change() {
        val name = settings_input_name.text.toString()
        val surname = settings_input_surname.text.toString()
        if (name.isEmpty()){
            showToast(getString(R.string.settings_toast_name_is_empty))
        } else {
            val fullname = "$name $surname"
            setNameToDatabase(fullname)
        }
    }
}
package com.example.dyp.fragments.settings

import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.fragments.base.BaseChangeFragment
import com.example.dyp.utilits.*
```

```
import kotlinx.android.synthetic.main.fragment_change_username.*
import java.util.*



class ChangeUsernameFragment :
BaseChangeFragment(R.layout.fragment_change_username) {

    lateinit var mNewUsername: String

    override fun onResume() {
        super.onResume()
        settings_input_username.setText(USER.username)
    }

    override fun change() {
        mNewUsername =
settings_input_username.text.toString().toLowerCase(Locale.getDefault())
        if (mNewUsername.isEmpty()){
            showToast("Поле пусте")
        } else {
            REF_DATABASE_ROOT.child(
                NODE_USERNAMES
            ).addListenerForSingleValueEvent(ValueEventListener{
                    if (it.hasChild(mNewUsername)){
                        showToast("Такий користувач уже існує")
                    } else{
                        changeUsername()
                    }
```

```
            })

        }
    }

    private fun changeUsername() {

REF_DATABASE_ROOT.child(NODE_USERNAMES).child(mNewUsername).setValue(
            CURRENT_UID
        )
            .addOnCompleteListener {
                if (it.isSuccessful){
                    updateCurrentUsername(mNewUsername)
                }
            }
    }




}
package com.example.dyp.fragments.settings

import android.app.Activity.RESULT_OK
import android.content.Intent
import android.view.Menu
import android.view.MenuInflater
```

```
import android.view.MenuItem
import com.example.dyp.R
import com.example.dyp.db.*
import com.example.dyp.fragments.base.BaseFragment
import com.example.dyp.utilits.*
import com.theartofdev.edmodo.cropper.CropImage
import com.theartofdev.edmodo.cropper.CropImageView
import kotlinx.android.synthetic.main.fragment_settings.*



class SettingsFragment : BaseFragment(R.layout.fragment_settings) {

    override fun onResume() {
        super.onResume()
        APP_ACTIVITY.title = "Налаштування"
        setHasOptionsMenu(true)
        initFields()
    }

    private fun initFields() {
        settings_bio.text = USER.bio
        settings_full_name.text = USER.fullname
        settings_phone_number.text = USER.phone
        settings_status.text = USER.state
        settings_username.text = USER.username
        settings_btn_change_username.setOnClickListener {
replaceFragment(ChangeUsernameFragment()) }
```

```
        settings_btn_change_bio.setOnClickListener {
replaceFragment(ChangeBioFragment()) }
        settings_change_photo.setOnClickListener { changePhotoUser() }
        settings_user_photo.downloadAndSetImage(USER.photoUrl)
    }

    private fun changePhotoUser() {
        CropImage.activity()
            .setAspectRatio(1, 1)
            .setRequestedSize(250, 250)
            .setCropShape(CropImageView.CropShape.OVAL)
            .start(APP_ACTIVITY,this)
    }


    override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
        activity?.menuInflater?.inflate(R.menu.settings_action_menu, menu)
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.settings_menu_exit -> {
                AppStates.updateState(AppStates.OFFLINE)
                AUTH.signOut()
                restartActivity()
            }
            R.id.settings_menu_change_name ->
replaceFragment(ChangeNameFragment())
        }
```

```
        return true
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE
            && resultCode == RESULT_OK && data != null
        ) {

            val uri = CropImage.getActivityResult(data).uri
            val path = REF_STORAGE_ROOT.child(
                PROFILE_IMAGE
            )
                .child(CURRENT_UID)
            loadfiletoDB(uri, path) {
                getUrlFromDB(path) {
                    loadUrltoDB(it) {
                        settings_user_photo.downloadAndSetImage(it)
                        showToast(getString(R.string.toast_data_update))
                        USER.photoUrl = it
                        APP_ACTIVITY.mAppDrawer.updHeader()
                    }
                }
            }
        }
    }

}
```

```
package com.example.dyp.utilits

import android.content.Context
import android.content.Intent
import android.net.Uri
import android.provider.ContactsContract
import android.provider.OpenableColumns
import android.view.inputmethod.InputMethodManager
import android.widget.ImageView
import android.widget.Toast
import androidx.fragment.app.Fragment
import com.example.dyp.MainActivity
import com.example.dyp.R
import com.example.dyp.db.updatePhonesToDB
import com.example.dyp.models.GeneralModel
import com.squareup.picasso.Picasso
import java.lang.Exception
import java.text.SimpleDateFormat
import java.util.*



fun showToast(message: String) {

    Toast.makeText(APP_ACTIVITY, message, Toast.LENGTH_SHORT).show()
}

fun restartActivity() {
```

```
    val intent = Intent(APP_ACTIVITY, MainActivity::class.java)
    APP_ACTIVITY.startActivity(intent)
    APP_ACTIVITY.finish()
}

fun replaceFragment(fragment: Fragment, addStack: Boolean = true) {

    if (addStack) {
        APP_ACTIVITY.supportFragmentManager.beginTransaction()
            .addToBackStack(null)
            .replace(
                R.id.data_container,
                fragment
            ).commit()
    } else {
        APP_ACTIVITY.supportFragmentManager.beginTransaction()
            .replace(
                R.id.data_container,
                fragment
            ).commit()
    }

}


fun Keyboard() {
```

```
    val imm: InputMethodManager =
APP_ACTIVITY.getSystemService(Context.INPUT_METHOD_SERVICE)
            as InputMethodManager

imm.hideSoftInputFromWindow(APP_ACTIVITY.window.decorView.windowToke
n, 0)
}

fun ImageView.downloadAndSetImage(url: String) {

    Picasso.get()
        .load(url)
        .fit()
        .placeholder(R.drawable.default_photo)
        .into(this)
}

fun initializeContacts() {

    if (checkPermission(READ_CONTACTS)) {
        var arrayContacts = arrayListOf<GeneralModel>()
        val cursor = APP_ACTIVITY.contentResolver.query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            null,
            null,
            null,
            null
        )
        cursor?.let {
```

```
            while (it.moveToNext()) {

                val fullName =
it.getString(it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME))
                val phone =
it.getString(it.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER))
                val newModel = GeneralModel()
                newModel.fullname = fullName
                newModel.phone = phone.replace(Regex("[\\s,-]"), "")
                arrayContacts.add(newModel)
            }
        }
        cursor?.close()
        updatePhonesToDB(arrayContacts)
    }
}

fun String.asTime(): String {
    val time = Date(this.toLong())
    val timeFormat = SimpleDateFormat("HH:mm", Locale.getDefault())
    return timeFormat.format(time)
}

fun getFilenameFromUri(uri: Uri): String {
    var result = ""
    val cursor = APP_ACTIVITY.contentResolver.query(uri, null, null, null, null)
```

```
    try {
        if (cursor != null && cursor.moveToFirst()) {
            result =
cursor.getString(cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME))
        }
    } catch (e: Exception) {
        showToast(e.message.toString())
    } finally {
        cursor?.close()
        return result
    }
}

fun getPlurals(count:Int) = APP_ACTIVITY.resources.getQuantityString(
    R.plurals.count_members,count,count
)
package com.example.dyp.message_views.message_adapter

import android.view.View
import android.widget.TextView
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.db.CURRENT_UID
import com.example.dyp.ui.message_views.message_view.MessageView
import com.example.dyp.utilits.asTime
import kotlinx.android.synthetic.main.message_item_text.view.*

class AdapterTextMessage(view: View) : RecyclerView.ViewHolder(view),
MessageAdapters {
```

```
private  val blocUserMessage: ConstraintLayout = view.bloc_user_message
private  val UserMessage: TextView = view.user_message
private  val UserMessageTime: TextView = view.user_message_time
private  val blocRecMessage: ConstraintLayout = view.bloc_rec_message
private  val RecMessage: TextView = view.rec_message
private  val RecMessageTime: TextView = view.rec_message_time


override fun drawMessage(view: MessageView) {
    if (view.from == CURRENT_UID) {
        blocUserMessage.visibility = View.VISIBLE
        blocRecMessage.visibility = View.GONE
        UserMessage.text = view.text
        UserMessageTime.text =
            view.timeStamp.asTime()
    } else {
        blocUserMessage.visibility = View.GONE
        blocRecMessage.visibility = View.VISIBLE
        RecMessage.text = view.text
        RecMessageTime.text =
            view.timeStamp.asTime()
    }
}

override fun onAttach(view: MessageView) {
}

override fun onDetach() {
```

```
    }
}
package com.example.dyp.message_views.message_adapter

import android.view.View
import android.widget.ImageView
import android.widget.TextView
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.db.CURRENT_UID
import com.example.dyp.ui.message_views.message_view.MessageView
import com.example.dyp.utilits.VoicePlayer
import com.example.dyp.utilits.asTime
import kotlinx.android.synthetic.main.message_item_voice.view.*

class AdapterVoiceMessage(view: View) : RecyclerView.ViewHolder(view),
MessageAdapters {

    private val mAppVoicePlayer = VoicePlayer()

    private val blocRecVoiceMessage: ConstraintLayout =
view.bloc_rec_voice_message
    private val blocUserVoiceMessage: ConstraintLayout =
view.bloc_user_voice_message
    private val VoiceMessageTime: TextView = view.voice_message_time
    private val RecVoiceMessageTime: TextView = view.rec_voice_message_time



    private val RecBtnPlay: ImageView = view.rec_btn_play
```

```
private val RecBtnStop: ImageView = view.rec_btn_stop

private val UserBtnPlay: ImageView = view.user_btn_play
private val UserBtnStop: ImageView = view.user_btn_stop

override fun onAttach(view: MessageView) {
    mAppVoicePlayer.init()
    if (view.from == CURRENT_UID) {
        UserBtnPlay.setOnClickListener {
            UserBtnPlay.visibility = View.GONE
            UserBtnStop.visibility = View.VISIBLE
            UserBtnStop.setOnClickListener {
                stop {
                    UserBtnStop.setOnClickListener(null)
                    UserBtnPlay.visibility = View.VISIBLE
                    UserBtnStop.visibility = View.GONE
                }
            }
            go(view) {
                UserBtnPlay.visibility = View.VISIBLE
                UserBtnStop.visibility = View.GONE
            }
        }
    } else {
        RecBtnPlay.setOnClickListener {
            RecBtnPlay.visibility = View.GONE
            RecBtnStop.visibility = View.VISIBLE
            RecBtnStop.setOnClickListener {
```

```
            stop {
                RecBtnStop.setOnClickListener(null)
                RecBtnPlay.visibility = View.VISIBLE
                RecBtnStop.visibility = View.GONE
            }
        }
        go(view) {
            RecBtnPlay.visibility = View.VISIBLE
            RecBtnStop.visibility = View.GONE
        }
    }
  }
}
override fun drawMessage(view: MessageView) {
  if (view.from == CURRENT_UID) {
    blocRecVoiceMessage.visibility = View.GONE
    blocUserVoiceMessage.visibility = View.VISIBLE
    VoiceMessageTime.text =
      view.timeStamp.asTime()
  } else {
    blocRecVoiceMessage.visibility = View.VISIBLE
    blocUserVoiceMessage.visibility = View.GONE
    RecVoiceMessageTime.text =
      view.timeStamp.asTime()
  }
}
```

```
    private fun go(
        view: MessageView,
        function: () -> Unit
    ) {
        mAppVoicePlayer.play(view.id, view.fileUrl) {
            function()
        }
    }

    fun stop(function: () -> Unit) {
        mAppVoicePlayer.stop {
            function()
        }
    }

    override fun onDetach() {
        RecBtnPlay.setOnClickListener(null)
        UserBtnPlay.setOnClickListener(null)
        mAppVoicePlayer.release()
    }

}package com.example.dyp.message_views.message_adapter

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.R
```

```
import com.example.dyp.ui.message_views.message_view.MessageView

class AppAdapter {
    companion object {
        fun getHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
            return when (viewType) {
                MessageView.MESSAGE_IMAGE -> {
                    val view = LayoutInflater.from(parent.context)
                        .inflate(R.layout.message_item_image, parent, false)
                    AdapterImageMessage(view)
                }

                MessageView.MESSAGE_VOICE -> {
                    val view = LayoutInflater.from(parent.context)
                        .inflate(R.layout.message_item_voice, parent, false)
                    AdapterVoiceMessage(view)
                }

                MessageView.MESSAGE_FILE -> {
                    val view = LayoutInflater.from(parent.context)
                        .inflate(R.layout.message_item_file, parent, false)
                    AdapterFileMessage(view)
                }

                else ->{
                    val view = LayoutInflater.from(parent.context)
                        .inflate(R.layout.message_item_text, parent, false)
                    AdapterTextMessage(view)
```

```
            }
        }
    }
}package com.example.dyp.message_views.message_adapter

import com.example.dyp.ui.message_views.message_view.MessageView

interface MessageAdapters {
    fun drawMessage(view: MessageView)
    fun onAttach(view: MessageView)
    fun onDetach()
}package com.example.dyp.models

data class GeneralModel(
    val id: String = "",
    var username: String = "",
    var bio: String = "",
    var fullname: String = "",
    var state: String = "",
    var phone: String = "",
    var photoUrl: String = "empty",

    var text: String = "",
    var type: String = "",
    var from: String = "",
    var timeStamp: Any = "",
    var fileUrl: String = "empty",
```

```
    var lastMessage:String = "",
    var choice:Boolean = false
) {
    override fun equals(other: Any?): Boolean {
        return (other as GeneralModel).id == id
    }
}package com.example.dyp.models

data class UserModel(
    val id: String = "",
    var username: String = "",
    var bio: String = "",
    var fullname: String = "",
    var state: String = "",
    var phone: String = "",
    var photoUrl: String = "empty"
)package com.example.dyp.utilits

import com.example.dyp.MainActivity

lateinit var APP_ACTIVITY:MainActivity
const val TYPE_MESSAGE_IMAGE = "image"
const val TYPE_MESSAGE_VOICE ="voice"
const val TYPE_MESSAGE_FILE = "file"
const val TYPE_CHAT ="chat"
const val TYPE_GROUP ="group"
const val PICK_FILE_REQUEST_CODE = 301
```

```
package com.example.dyp.utilits

import android.Manifest
import android.content.pm.PackageManager
import android.os.Build
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat


const val READ_CONTACTS = Manifest.permission.READ_CONTACTS
const val RECORD_AUDIO = Manifest.permission.RECORD_AUDIO
const val WRITE_FILES = Manifest.permission.WRITE_EXTERNAL_STORAGE

const val PERMISSION_REQUEST = 200

fun checkPermission(permission: String): Boolean {

   return if (Build.VERSION.SDK_INT >= 23
      && ContextCompat.checkSelfPermission(
         APP_ACTIVITY,
         permission
      ) != PackageManager.PERMISSION_GRANTED
   ) {
      ActivityCompat.requestPermissions(APP_ACTIVITY, arrayOf(permission),
PERMISSION_REQUEST)
      false
   } else true
}package com.example.dyp.utilits
```

```
import com.example.dyp.db.*

enum class AppStates(val state:String) {


    ONLINE("в мережі"),
    OFFLINE("був нещодавно"),
    TYPING("друкує...");

    companion object{
        fun updateState(appStates: AppStates){

            if (AUTH.currentUser!=null){
                REF_DATABASE_ROOT.child(
                    USERS
                ).child(CURRENT_UID).child(
                    CHILD_STATE
                )
                    .setValue(appStates.state)
                    .addOnSuccessListener { USER.state = appStates.state }
                    .addOnFailureListener { showToast(it.message.toString()) }
            }
        }
    }
}package com.example.dyp.utilits

import com.google.firebase.database.ChildEventListener
```

```
import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError

class ChildEventListener (val onSuccess:(DataSnapshot) -> Unit) :
    ChildEventListener {
    override fun onCancelled(p0: DatabaseError) {
    }
    override fun onChildMoved(p0: DataSnapshot, p1: String?) {
    }
    override fun onChildChanged(p0: DataSnapshot, p1: String?) {
    }
    override fun onChildAdded(p0: DataSnapshot, p1: String?) {
        onSuccess(p0)
    }
    override fun onChildRemoved(p0: DataSnapshot) {
    }



}package com.example.dyp.utilits

import androidx.recyclerview.widget.DiffUtil
import com.example.dyp.models.GeneralModel

class DiffUtilCalback(
    private val oldList: List<GeneralModel>,
    private val newList: List<GeneralModel>
):DiffUtil.Callback() {
```

```
    override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int):
Boolean =
        oldList[oldItemPosition].timeStamp == newList[newItemPosition].timeStamp


    override fun getOldListSize(): Int = oldList.size


    override fun getNewListSize(): Int = newList.size


    override fun areContentsTheSame(oldItemPosition: Int, newItemPosition: Int):
Boolean =
        oldList[oldItemPosition] == newList[newItemPosition]
}package com.example.dyp.utilits


import android.content.Context
import android.content.Intent
import android.net.Uri
import android.provider.ContactsContract
import android.provider.OpenableColumns
import android.view.inputmethod.InputMethodManager
import android.widget.ImageView
import android.widget.Toast
import androidx.fragment.app.Fragment
import com.example.dyp.MainActivity
import com.example.dyp.R
import com.example.dyp.db.updatePhonesToDB
import com.example.dyp.models.GeneralModel
import com.squareup.picasso.Picasso
import java.lang.Exception
import java.text.SimpleDateFormat
```

```
import java.util.*



fun showToast(message: String) {

    Toast.makeText(APP_ACTIVITY, message, Toast.LENGTH_SHORT).show()
}

fun restartActivity() {

    val intent = Intent(APP_ACTIVITY, MainActivity::class.java)
    APP_ACTIVITY.startActivity(intent)
    APP_ACTIVITY.finish()
}

fun replaceFragment(fragment: Fragment, addStack: Boolean = true) {

    if (addStack) {
        APP_ACTIVITY.supportFragmentManager.beginTransaction()
            .addToBackStack(null)
            .replace(
                R.id.data_container,
                fragment
            ).commit()
    } else {
        APP_ACTIVITY.supportFragmentManager.beginTransaction()
            .replace(
```

```
            R.id.data_container,
            fragment
        ).commit()
    }

}


fun Keyboard() {

    val imm: InputMethodManager =
APP_ACTIVITY.getSystemService(Context.INPUT_METHOD_SERVICE)
        as InputMethodManager

imm.hideSoftInputFromWindow(APP_ACTIVITY.window.decorView.windowToke
n, 0)
}

fun ImageView.downloadAndSetImage(url: String) {

    Picasso.get()
        .load(url)
        .fit()
        .placeholder(R.drawable.default_photo)
        .into(this)
}

fun initializeContacts() {
```

```
    if (checkPermission(READ_CONTACTS)) {
        var arrayContacts = arrayListOf<GeneralModel>()
        val cursor = APP_ACTIVITY.contentResolver.query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            null,
            null,
            null,
            null
        )
        cursor?.let {
            while (it.moveToNext()) {

                val fullName =

it.getString(it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME))
                val phone =

it.getString(it.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMB
ER))
                val newModel = GeneralModel()
                newModel.fullname = fullName
                newModel.phone = phone.replace(Regex("[\\s,-]"), "")
                arrayContacts.add(newModel)
            }
        }
        cursor?.close()
        updatePhonesToDB(arrayContacts)
    }
}
```

```
fun String.asTime(): String {
    val time = Date(this.toLong())
    val timeFormat = SimpleDateFormat("HH:mm", Locale.getDefault())
    return timeFormat.format(time)
}

fun getFilenameFromUri(uri: Uri): String {
    var result = ""
    val cursor = APP_ACTIVITY.contentResolver.query(uri, null, null, null, null)
    try {
        if (cursor != null && cursor.moveToFirst()) {
            result =
cursor.getString(cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME))
        }
    } catch (e: Exception) {
        showToast(e.message.toString())
    } finally {
        cursor?.close()
        return result
    }
}

fun getPlurals(count:Int) = APP_ACTIVITY.resources.getQuantityString(
    R.plurals.count_members,count,count
)package com.example.dyp.utilits

import android.text.Editable
import android.text.TextWatcher
```

```
class TextWatcher(val onSuccess: (Editable?) -> Unit) :
   TextWatcher {

   override fun afterTextChanged(s: Editable?) {
      onSuccess(s)
   }

   override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int)
{}

   override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int)
{}
}package com.example.dyp.utilits

import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener



class ValueEventListener (val onSuccess:(DataSnapshot) -> Unit) :
   ValueEventListener {
   override fun onCancelled(p0: DatabaseError) {

   }

   override fun onDataChange(p0: DataSnapshot) {
```

```
        onSuccess(p0)
    }

}package com.example.dyp.utilits

import com.google.firebase.database.DataSnapshot
import com.google.firebase.database.DatabaseError
import com.google.firebase.database.ValueEventListener



class ValueEventListener (val onSuccess:(DataSnapshot) -> Unit) :
    ValueEventListener {
    override fun onCancelled(p0: DatabaseError) {

    }

    override fun onDataChange(p0: DataSnapshot) {
        onSuccess(p0)
    }

}package com.example.dyp.utilits

import android.media.MediaRecorder
import java.io.File

class VoiceRecorder {
```

```
private val messageMediaRecorder = MediaRecorder()
private lateinit var messageFile: File
private lateinit var mMessageKey: String

fun startRecord(messageKey: String) {
    try {
        mMessageKey = messageKey
        createFileForRecord()
        prepareMediaRecorder()
        messageMediaRecorder.start()
    } catch (e: Exception) {
        showToast(e.message.toString())
    }

}

private fun prepareMediaRecorder() {
    messageMediaRecorder.apply {
        reset()
        setAudioSource(MediaRecorder.AudioSource.DEFAULT)
        setOutputFormat(MediaRecorder.OutputFormat.DEFAULT)
        setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT)
        setOutputFile(messageFile.absolutePath)
        prepare()
    }

}
```

```
    private fun createFileForRecord() {
        messageFile = File(APP_ACTIVITY.filesDir, mMessageKey)
        messageFile.createNewFile()
    }

    fun stopRecord(onSuccess: (file: File, messageKey: String) -> Unit) {
        try {
            messageMediaRecorder.stop()
            onSuccess(messageFile, mMessageKey)
        } catch (e: Exception) {
            showToast(e.message.toString())
            messageFile.delete()
        }
    }

    fun releaseRecorder() {
        try {
            messageMediaRecorder.release()
        } catch (e: Exception) {
            showToast(e.message.toString())
        }
    }

}package com.example.dyp

import android.content.pm.PackageManager
import android.os.Bundle
```

```
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.core.content.ContextCompat
import com.example.dyp.db.AUTH
import com.example.dyp.db.initializationFirebase
import com.example.dyp.db.initUser
import com.example.dyp.databinding.ActivityMainBinding
import com.example.dyp.fragments.main_list.MainListFragment
import com.example.dyp.fragments.register.EnterPhoneNumberFragment
import com.example.dyp.Drawers.AppDrawer
import com.example.dyp.utilits.*
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch


class MainActivity : AppCompatActivity() {

    private lateinit var mBinding: ActivityMainBinding
    lateinit var mAppDrawer: AppDrawer
    lateinit var mToolbar: Toolbar
    override fun onCreate(savedInstanceState: Bundle?) {
var adapter: MyAdapter? = null
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        nav_view.setNavigationItemSelectedListener (this)
```

```
        var list = ArrayList<itemList>()
        list.addAll(fillArras(resources.getStringArray(R.array.all),
resources.getStringArray(R.array.all_content),getImageId(R.array.all_image_array),getImageId2(R.array.all_navi_image_array)))
        rcView.hasFixedSize()
        rcView.layoutManager = LinearLayoutManager(this)
        adapter = MyAdapter(list, this)
        rcView.adapter = adapter
    }

    override fun onNavigationItemSelected(item: MenuItem): Boolean {

        when(item.itemId){

            R.id.id_search ->
            {
                Toast.makeText(this,"Id search",Toast.LENGTH_SHORT).show()
adapter?.navAdapterUpdate(searcharrays(resources.getStringArray(R.array.search),
resources.getStringArray(R.array.search_content),getImageId(R.array.search_image_array),getImageId2(R.array.search_navi_image_array)))

            }
            R.id.id_all ->
            {
                Toast.makeText(this,"Id recommend",Toast.LENGTH_SHORT).show()
```

```
adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.all),

                    resources.getStringArray(R.array.all_content),
getImageId(R.array.all_image_array), getImageId2(R.array.all_navi_image_array)))

        }
        R.id.id_recommend ->
        {
            Toast.makeText(this,"Id recommend",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.recommend),

resources.getStringArray(R.array.recommend_content),getImageId(R.array.recommend_image_array),getImageId2(R.array.recommend_navi_image_array)))

        }
        R.id.id_park ->
        {
            Toast.makeText(this,"Id park",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.park),

resources.getStringArray(R.array.park_content),getImageId(R.array.park_image_array),getImageId2(R.array.park_navi_image_array)))

        }



        R.id.id_beaches ->
        {
```

```
            Toast.makeText(this,"Id history",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.beaches),

resources.getStringArray(R.array.beaches_content),getImageId(R.array.beaches_image_array),getImageId2(R.array.beaches_navi_image_array)))


        }
        R.id.id_archi ->
        {
            Toast.makeText(this,"Id archi",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.archi),

resources.getStringArray(R.array.archi_content),getImageId(R.array.archi_image_array),getImageId2(R.array.archi_navi_image_array)))


        }
        R.id.id_teatr ->
        {
            Toast.makeText(this,"Id teatr",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.teatr),

resources.getStringArray(R.array.teatr_content),getImageId(R.array.teatr_image_array),getImageId2(R.array.teatr_navi_image_array)))


        }
        R.id.id_museum ->
        {
            Toast.makeText(this,"Id museum",Toast.LENGTH_SHORT).show()
```

```
adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.museum),

resources.getStringArray(R.array.museum_content),getImageId(R.array.museum_im
age_array),getImageId2(R.array.museum_navi_image_array)))


        }
        R.id.id_derzh ->
        {
            Toast.makeText(this,"Id derzh",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.derzh),

resources.getStringArray(R.array.derzh_content),getImageId(R.array.derzh_image_ar
ray),getImageId2(R.array.derzh_navi_image_array)))


        }
        R.id.id_school ->
        {
            Toast.makeText(this,"Id school",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.school),

resources.getStringArray(R.array.school_content),getImageId(R.array.school_image_
array),getImageId2(R.array.school_navi_image_array)))


        }
        R.id.id_dyt_sad ->
        {
            Toast.makeText(this,"Id dyt_sad",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.dyt_sad),
```

```
resources.getStringArray(R.array.dyt_sad_content),getImageId(R.array.dyt_sad_ima
ge_array),getImageId2(R.array.dyt_sad_navi_image_array)))


        }
        R.id.id_med ->
        {
            Toast.makeText(this,"Id med",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.med),

resources.getStringArray(R.array.med_content),getImageId(R.array.med_image_arra
y),getImageId2(R.array.med_navi_image_array)))


        }
        R.id.id_metro ->
        {
            Toast.makeText(this,"Id metro",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.metro),

resources.getStringArray(R.array.metro_content),getImageId(R.array.metro_image_a
rray),getImageId2(R.array.metro_navi_image_array)))


        }
        R.id.id_aero ->
        {
            Toast.makeText(this,"Id aero",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.aero),
```

```
resources.getStringArray(R.array.aero_content),getImageId(R.array.aero_image_array),getImageId2(R.array.aero_navi_image_array)))


            }
            R.id.id_avto ->
            {
                Toast.makeText(this,"Id avto",Toast.LENGTH_SHORT).show()

adapter?.navAdapterUpdate(fillArras(resources.getStringArray(R.array.avto),

resources.getStringArray(R.array.avto_content),getImageId(R.array.avto_image_array),getImageId2(R.array.avto_navi_image_array)))


            }
        }
        drawerLayout.closeDrawer(GravityCompat.START)


        return true
    }
    fun fillArras(titleArray:Array<String>,contentArray:Array<String>,imageArray:IntArray, imageArray2:IntArray):List<itemList>
    {
        var itemListArray = ArrayList<itemList>()
        for(n in 0..titleArray.size - 1)
        {
            var itemList = itemList(imageArray[n], titleArray[n], contentArray[n], imageArray2[n])
            itemListArray.add(itemList)
        }
```

```
        return itemListArray
    }
    fun searcharrays(titleArray:Array<String>,contentArray:Array<String>,imageArray:IntArray, imageArray2:IntArray):List<itemList>
    {
        var itemListArray = ArrayList<itemList>()
        for(n in 0..titleArray.size - 1)
        {

         if(titleArray[n].contains(search_text_view.text.toString(), true)==true) {
        //  if(titleArray[n].contains("Па")==true) {
              var itemList = itemList(imageArray[n], titleArray[n], contentArray[n], imageArray2[n])

                itemListArray.add(itemList)

           }
        }
        return itemListArray
    }

    fun getImageId(imageArrayId:Int):IntArray
    {
        var iArray:TypedArray = resources.obtainTypedArray(imageArrayId)
        val counter = iArray.length()
        val id_img = IntArray(counter)
        for(i in id_img.indices)
        {
```

```
            id_img[i] = iArray.getResourceId(i,0)
        }
        iArray.recycle()
        return id_img
    }
    fun getImageId2(imageArray2Id:Int):IntArray
    {
        var iArray2:TypedArray = resources.obtainTypedArray(imageArray2Id)
        val counter2 = iArray2.length()
        val id_img2 = IntArray(counter2)
        for(j in id_img2.indices)
        {
            id_img2[j] = iArray2.getResourceId(j,0)
        }
        iArray2.recycle()
        return id_img2
    }


}
        super.onCreate(savedInstanceState)
        mBinding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(mBinding.root)
        APP_ACTIVITY = this
        initializationFirebase()
        initUser {
            CoroutineScope(Dispatchers.IO).launch {
                initializeContacts()
```

```
        }
        initFields()
        initFunc()
    }

}



private fun initFunc() {

    setSupportActionBar(mToolbar)
    if (AUTH.currentUser != null) {
        mAppDrawer.create()
        replaceFragment(MainListFragment(), false)
    } else {
        replaceFragment(EnterPhoneNumberFragment(),false)
    }
}

private fun initFields() {

    mToolbar = mBinding.mainToolbar
    mAppDrawer = AppDrawer()
}

override fun onStart() {
    super.onStart()
```

```
        AppStates.updateState(AppStates.ONLINE)
    }

    override fun onStop() {
        super.onStop()
        AppStates.updateState(AppStates.OFFLINE)
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray
    ) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        if (ContextCompat.checkSelfPermission(APP_ACTIVITY,
READ_CONTACTS)==PackageManager.PERMISSION_GRANTED){
            initializeContacts()
        }
    }
}
import com.example.dyp.models.UserModel
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.DatabaseReference
import com.google.firebase.storage.StorageReference

lateinit var AUTH: FirebaseAuth
lateinit var CURRENT_UID: String
lateinit var REF_DATABASE_ROOT: DatabaseReference
lateinit var REF_STORAGE_ROOT: StorageReference
```

```
lateinit var USER: UserModel
const val TEXT_TYPE = "text"
const val USERS = "users"
const val NODE_MESSAGES = "messages"
const val NODE_USERNAMES = "usernames"
const val NODE_PHONES = "phones"
const val NODE_PHONES_CONTACTS = "phones_contacts"
const val PROFILE_IMAGE = "profile_image"
const val FILE_FOLDER = "messages_files"
const val CHILD_ID = "id"
const val CHILD_PHONE = "phone"
const val CHILD_USERNAME = "username"
const val CHILD_FULLNAME = "fullname"
const val CHILD_INFO = "info"
const val CHILD_PHOTO_URL = "photoUrl"
const val CHILD_STATE = "state"
const val CHILD_TEXT = "text"
const val CHILD_TYPE = "type"
const val CHILD_FROM = "from"
const val CHILD_TIMESTAMP = "timeStamp"
const val CHILD_FILE_URL = "fileUrl"
const val NODE_MAIN_LIST = "main_list"
const val NODE_GROUPS = "groups"
const val NODE_MEMBERS = "members"
const val FOLDER_GROUPS_IMAGE = "groups_image"
const val USER_CREATOR = "creator"
const val USER_MEMBER = "member"package com.example.dyp.utilits
package com.example.dyp.message_views.message_adapter
```

```
import android.os.Environment
import android.view.View
import android.widget.ImageView
import android.widget.ProgressBar
import android.widget.TextView
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.db.CURRENT_UID
import com.example.dyp.db.downloadFilewithDB
import com.example.dyp.ui.message_views.message_view.MessageView
import com.example.dyp.utilits.WRITE_FILES
import com.example.dyp.utilits.asTime
import com.example.dyp.utilits.checkPermission
import com.example.dyp.utilits.showToast
import kotlinx.android.synthetic.main.message_item_file.view.*
import java.io.File
import java.lang.Exception
    var arraylistR = arrayList
    var rContext = context


    class ViewHolder(view: View): RecyclerView.ViewHolder(view) {
        val infoTitle = view.findViewById<TextView>(R.id.infoTitle)
        val infoContent = view.findViewById<TextView>(R.id.infoContent)
        val image = view.findViewById<ImageView>(R.id.image)
        val image2 = view.findViewById<ImageView>(R.id.image2)
```

```
        fun bind(listItem: itemList, context: Context)
        {

            infoTitle.text = listItem.infoTitle
            var textContent = listItem.contentInfo.substring(0,50) + "..."
            infoContent.text = textContent
            image.setImageResource(listItem.images_id)

            itemView.setOnClickListener(){
                Toast.makeText(context,"Pressed :
${infoTitle.text}",Toast.LENGTH_SHORT).show()
                val i = Intent(context,
                    ContentActivity::class.java).apply {
                    putExtra("title",infoTitle.text.toString())
                    putExtra("content",listItem.contentInfo)
                    putExtra("images",listItem.images_id)
                    putExtra("images2",listItem.images_id2)
                }
                context.startActivity(i)









            }
        }

    }
```

```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val createVH = LayoutInflater.from(rContext)
        return ViewHolder(
            createVH.inflate(
                R.layout.item_layout,
                parent,
                false
            )
        )
    }

    override fun getItemCount(): Int {
        return arraylistR.size
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        var listItem = arraylistR.get(position)
        holder.bind(listItem,rContext)
    }

    fun navAdapterUpdate(arrayList : List<itemList>)
    {
        arraylistR.clear()
        arraylistR.addAll(arrayList)
        notifyDataSetChanged()
    }
}
```

```
class AdapterFileMessage(view: View) : RecyclerView.ViewHolder(view),
MessageAdapters {

    private val recievedmessage: ConstraintLayout = view.bloc_rec_message
    private val usermessage: ConstraintLayout = view.bloc_user_message
    private val fileMessageTime: TextView = view.file_message_time
    private val recFiliMessageTime: TextView = view.rec_file_message_time

    private val chatUserFile:TextView = view.chat_user_file
    private val chatUserButtonDown:ImageView = view.chat_user_button
    private val chatUserProgress:ProgressBar = view.chat_user_progress


    private val chatRecFile:TextView = view.chat_rec_file
    private val chatRecButtonDown:ImageView = view.chat_rec_btn_down
    private val chatRecProgress:ProgressBar = view.chat_rec_progress


    override fun onAttach(view: MessageView) {
       if (view.from == CURRENT_UID)chatUserButtonDown.setOnClickListener {
clickToBtnFile(view) }
       else chatRecButtonDown.setOnClickListener {clickToBtnFile(view) }
    }

    private fun clickToBtnFile(view: MessageView) {
        if (view.from == CURRENT_UID){
            chatUserButtonDown.visibility = View.INVISIBLE
            chatUserProgress.visibility = View.VISIBLE
        } else {
```

```
            chatRecButtonDown.visibility = View.INVISIBLE
            chatRecProgress.visibility = View.VISIBLE
        }

        val file = File(
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS),
            view.text
        )

        try {
            if (checkPermission(WRITE_FILES)){
                file.createNewFile()
                downloadFilewithDB(file,view.fileUrl){
                    if (view.from == CURRENT_UID){
                        chatUserButtonDown.visibility = View.VISIBLE
                        chatUserProgress.visibility = View.INVISIBLE
                    } else {
                        chatRecButtonDown.visibility = View.VISIBLE
                        chatRecProgress.visibility = View.INVISIBLE
                    }
                }
            }
        }catch (e:Exception){
            showToast(e.message.toString())
        }
    }
```

```
    override fun drawMessage(view: MessageView) {
        if (view.from == CURRENT_UID) {
            recievedmessage.visibility = View.GONE
            usermessage.visibility = View.VISIBLE
            fileMessageTime.text = view.timeStamp.asTime()
            chatUserFile.text = view.text
        } else {
            recievedmessage.visibility = View.VISIBLE
            usermessage.visibility = View.GONE
            recFiliMessageTime.text = view.timeStamp.asTime()
            chatRecFile.text = view.text
        }
    }

    override fun onDetach() {
      chatUserButtonDown.setOnClickListener(null)
      chatRecButtonDown.setOnClickListener(null)
    }

} package com.example.dyp.utilits

import android.media.MediaRecorder
import java.io.File

class VoiceRecorder {


    private val messageMediaRecorder = MediaRecorder()
```

```
private lateinit var messageFile: File
private lateinit var mMessageKey: String

fun startRecord(messageKey: String) {
    try {
        mMessageKey = messageKey
        createFileForRecord()
        prepareMediaRecorder()
        messageMediaRecorder.start()
    } catch (e: Exception) {
        showToast(e.message.toString())
    }

}

private fun prepareMediaRecorder() {
    messageMediaRecorder.apply {
        reset()
        setAudioSource(MediaRecorder.AudioSource.DEFAULT)
        setOutputFormat(MediaRecorder.OutputFormat.DEFAULT)
        setAudioEncoder(MediaRecorder.AudioEncoder.DEFAULT)
        setOutputFile(messageFile.absolutePath)
        prepare()
    }

}

private fun createFileForRecord() {
```

```
        messageFile = File(APP_ACTIVITY.filesDir, mMessageKey)
        messageFile.createNewFile()
    }

    fun stopRecord(onSuccess: (file: File, messageKey: String) -> Unit) {
        try {
            messageMediaRecorder.stop()
            onSuccess(messageFile, mMessageKey)
        } catch (e: Exception) {
            showToast(e.message.toString())
            messageFile.delete()
        }
    }

    fun releaseRecorder() {
        try {
            messageMediaRecorder.release()
        } catch (e: Exception) {
            showToast(e.message.toString())
        }
    }

}
package com.example.dyp.message_views.message_adapter

import android.view.View
import android.widget.ImageView
import android.widget.TextView
```

```
import androidx.constraintlayout.widget.ConstraintLayout
import androidx.recyclerview.widget.RecyclerView
import com.example.dyp.db.CURRENT_UID
import com.example.dyp.ui.message_views.message_view.MessageView
import com.example.dyp.utilits.asTime
import com.example.dyp.utilits.downloadAndSetImage
import kotlinx.android.synthetic.main.message_item_image.view.*

class AdapterImageMessage(view: View) : RecyclerView.ViewHolder(view),
MessageAdapters {
    private  val ImageMessage: ConstraintLayout = view.image_message
    private  val Image_Message: ConstraintLayout = view.images_message
    private  val UserImage: ImageView = view.user_image
    private  val RecImage: ImageView = view.rec_image
    private  val UserImageMessageTime: TextView = view.user_image_message_time
    private  val RecImageMessageTime: TextView = view.rec_image_message_time


    override fun drawMessage(view: MessageView) {
        if (view.from == CURRENT_UID) {
            ImageMessage.visibility = View.GONE
            Image_Message.visibility = View.VISIBLE
            UserImage.downloadAndSetImage(view.fileUrl)
            UserImageMessageTime.text =
                view.timeStamp.asTime()
        } else {
            ImageMessage.visibility = View.VISIBLE
            Image_Message.visibility = View.GONE
            RecImage.downloadAndSetImage(view.fileUrl)
```

```
            RecImageMessageTime.text =
                view.timeStamp.asTime()
        }
    }

    override fun onAttach(view: MessageView) {
    }

    override fun onDetach() {
    }
}
import android.content.Context
import android.content.Intent
import android.net.Uri
import android.provider.ContactsContract
import android.provider.OpenableColumns
import android.view.inputmethod.InputMethodManager
import android.widget.ImageView
import android.widget.Toast
import androidx.fragment.app.Fragment
import com.example.dyp.MainActivity
import com.example.dyp.R
import com.example.dyp.db.updatePhonesToDB
import com.example.dyp.models.GeneralModel
import com.squareup.picasso.Picasso
import java.lang.Exception
import java.text.SimpleDateFormat
import java.util.*
```

```
fun showToast(message: String) {

    Toast.makeText(APP_ACTIVITY, message, Toast.LENGTH_SHORT).show()
}

fun restartActivity() {

    val intent = Intent(APP_ACTIVITY, MainActivity::class.java)
    APP_ACTIVITY.startActivity(intent)
    APP_ACTIVITY.finish()
}

fun replaceFragment(fragment: Fragment, addStack: Boolean = true) {

    if (addStack) {
        APP_ACTIVITY.supportFragmentManager.beginTransaction()
            .addToBackStack(null)
            .replace(
                R.id.data_container,
                fragment
            ).commit()
    } else {
        APP_ACTIVITY.supportFragmentManager.beginTransaction()
            .replace(
                R.id.data_container,
```

```
            fragment
        ).commit()
    }

}


fun Keyboard() {

    val imm: InputMethodManager =
APP_ACTIVITY.getSystemService(Context.INPUT_METHOD_SERVICE)
        as InputMethodManager

imm.hideSoftInputFromWindow(APP_ACTIVITY.window.decorView.windowToke
n, 0)
}

fun ImageView.downloadAndSetImage(url: String) {

    Picasso.get()
        .load(url)
        .fit()
        .placeholder(R.drawable.default_photo)
        .into(this)
}

fun initializeContacts() {

    if (checkPermission(READ_CONTACTS)) {
```

```
        var arrayContacts = arrayListOf<GeneralModel>()
        val cursor = APP_ACTIVITY.contentResolver.query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            null,
            null,
            null,
            null
        )
        cursor?.let {
            while (it.moveToNext()) {

                val fullName =

it.getString(it.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME))
                val phone =

it.getString(it.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMB
ER))
                val newModel = GeneralModel()
                newModel.fullname = fullName
                newModel.phone = phone.replace(Regex("[\\s,-]"), "")
                arrayContacts.add(newModel)
            }
        }
        cursor?.close()
        updatePhonesToDB(arrayContacts)
    }
}
```

```
fun String.asTime(): String {
    val time = Date(this.toLong())
    val timeFormat = SimpleDateFormat("HH:mm", Locale.getDefault())
    return timeFormat.format(time)
}

fun getFilenameFromUri(uri: Uri): String {
    var result = ""
    val cursor = APP_ACTIVITY.contentResolver.query(uri, null, null, null, null)
    try {
        if (cursor != null && cursor.moveToFirst()) {
            result =
cursor.getString(cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME))
        }
    } catch (e: Exception) {
        showToast(e.message.toString())
    } finally {
        cursor?.close()
        return result
    }
}

fun getPlurals(count:Int) = APP_ACTIVITY.resources.getQuantityString(
    R.plurals.count_members,count,count
)package com.example.dyp.fragments.base
```