

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ОФЛАЙН ЧАТ-ЗАСТОСУНКУ ІЗ ВИКОРИСТАННЯМ

ВЕБСОКЕТІВ

DEVELOPMENT OF AN OFFLINE CHAT APPLICATION

USING WEB SOCKETS

Виконав: студент(ка) 4 курсу, групи КНТ-138
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Снурніцин Б.Р.

(прізвище та ініціали)

Керівник Табунщик Г.В.

(прізвище та ініціали)

Рецензент Шитікова О.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Снурніцина Богдана Романовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка офлайн чат-застосунку із використанням вебсокетів. Development of an Offline Chat Application Using Web Sockets

керівник проєкту (роботи) Табунщик Галина Володимирівна, к.т.н., професор,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір та обґрунтування структури системи яка проєктується. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження застосунку.

Технічне завдання

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Табунщик Г.В., професор		
Нормоконтроль	Бєлова А.В., асистент		

7. Дата видачі завдання «04» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області та програмних засобів, що будуть використовуватись в роботі	2 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки	3 тиждень	Розділ 2
4	Розробка архітектури програми та логічної моделі системи.	4-5 тижні	Розділ 3
5	Розробка графічного інтерфейсу програми.	6-7 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	8 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї	8 тиждень	Додатки
8	Нормоконтроль та рецензування.	9 тиждень	
9	Захист роботи.	10 тиждень	

Студент(ка)

(підпис) Снурніцин Б.Р.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Табунщик Г.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
258 с., 93 рис., 2 табл., 3 дод., 17 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, ОФЛАЙН ЧАТ, KOTLIN,
WI-FI, SOCKET.

Об'єкт дослідження – програмні засоби для підтримки зв'язку та спілкування.

Предмет дослідження – застосунок призначений для комунікування без доступу до мережі інтернет.

Мета роботи – розробка Android – додатку із можливостями створення мережеских груп, підключення до такої ж створеної групи та підтримка комунікації зі всіма учасниками групи, з легким для розуміння інтерфейсом.

Матеріали, методи та технічні засоби: функціональне та об'єктно-орієнтоване програмування, мобільний пристрій з операційною системою Android, мова програмування Kotlin.

Результати. Створено програмний Android-застосунок, що дозволяє користувачам обмінюватись текстовими повідомленнями підключившись до однієї мережі Wi-Fi.

Висновки. Створено програмний Android-застосунок, що має змогу створювати групи, підключатись до груп та синхронізуватись між собою у діапазоні роботи мережі Wi-Fi без доступу до Інтернету із можливістю зберігання даних сесії.

Галузь використання – створення груп для спілкування між користувачами у межах діапазону роботи Wi-Fi у разі необхідності зв'язку, але відсутності або нестабільності Інтернет з'єднання.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 258 pages, 93 figures, 2 tables, 3 appendixes, 17 sources.

MOBILE APPLICATION, ANDROID, WI-FI, OFFLINE CHAT, KOTLIN, ANDROID STUDIO.

The object of research is software for communication.

Subject of research - an application designed to communicate without access to the Internet.

The purpose of the work is to develop an Android application with the ability to create network groups, connect to such created groups and support communication with all the members of the group, with an easy-to-understand interface.

Materials, methods and technical means: functional and object-oriented programming, mobile device with Android operating system, Kotlin programming language.

Results. An Android application has been created which allows users to exchange text messages by connecting to the same Wi-Fi network.

Conclusions. An Android application has been created which is able to create groups, connect to groups, and synchronize with each other within range of a Wi-Fi network without Internet access with the ability to store session data.

Usage scope – Creation of groups for communication with users within range of Wi-Fi in case of need to be connected, but the absence or instability of an Internet connection.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	7
Вступ	8
1 Аналіз предметної області	9
1.1 Знайомство з предметною областю	9
1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання	9
1.3 Висновки.....	14
2 Вибір та обґрунтування структури системи яка проектується	15
2.1 Вибір інструментарію розробки програмного забезпечення.....	15
2.2 Структура файлів Android застосунку	21
3 Основні рішення щодо реалізації компонентів системи	23
3.1 Реалізація архітектури та патернів програмування	23
3.2 Огляд компонентів та розроблених класів	34
3.3 Огляд візуального інтерфейсу застосунку	70
3.4 Огляд інтерактивних повідомлень.....	94
4 Експлуатація, тестування та експериментальне дослідження застосунку	98
4.1 Процес роботи застосунку	98
4.2 Тестування застосунку	100
4.3 Висновки за розділом 4.....	101
Висновки.....	102
Перелік посилань	104
Додаток А Технічне завдання	106
Додаток Б Текст програми.....	248
Додаток В Слайди презентації	248

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- IDE – Integrated Development Environment;
QR – quick response code;
ПЗ – програмне забезпечення;
IP – Internet Protocol.

ВСТУП

Спілкування завжди було невід'ємною частиною життя людини як у давнину, так і у наші дні. І здавалося б, у епоху Інтернету та прогресивних технологій питання доступного спілкування, зв'язку та обмін інформацією вже не може бути проблемою. Чи ні? На жаль, ця проблема буде переслідувати нас, допоки у кожному куточку планети не буде стояти мережева станція. Тож стає питання, як бути та як підтримувати зв'язок ?

Мета мобільного додатку – це надання можливості користувачам створювати групи для спілкування та обмінюватись інформацією без підключення до Інтернету, завдяки технології Wi-Fi. На жаль, це не вирішує проблему спілкування на відстані більше ніж 50-100 метрів, залежачи від потужності Wi-Fi модуля та інших параметрів [1]-[2]. Але тим не менш це вирішує чимало інших проблем у кризових ситуаціях. Уявіть ситуації, що ви йдете у метро, або у поїзді за містом, та потрібно зв'язатись із близькими або друзями, що знаходяться відносно далеко від вас – від кількох метрів до, скажемо, іншого вагону. Або ситуації, коли мережеві станції операторів зв'язку глушать навмисно, і є потреба бути на зв'язку з іншими людьми поблизу задля вчасного інформування. Не менш поширеними ситуаціями є вечірки, що влаштовуються далеко від зони обслуговування станціями. В усіх цих ситуаціях є одна мета – налагодження контакту з користувачами поблизу і усі ці проблеми вирішуються за допомогою «Juicy Wi-Fi Chat».

Окрім цього, основною перевагою додатку вважається приємне та унікальне стильове оформлення із зрозумілим інтерфейсом, що не потребує багато часу для початку роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Знайомство з предметною областю

Основні функції застосунку – це пошук груп із користувачами поблизу, можливість з'єднання з користувачами та обмін інформацією за допомогою текстових повідомлень. Зробивши дослідження на наявність аналогів, що вирішують такі ж проблеми, можна зазначити, що всі вони відрізняються як за інтерфейсом так і за функціоналом.

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

1.2.1 Застосунок «Briar»

Briar [3] – це застосунок для листування, що надає надійний засіб зв'язку. Інтерфейс додатку виглядає мінімалістично – меню, із вибором контактів, груп або форумів, поле налаштування та вихід з додатку. Також знизу є елементи налаштування типу підключення між контактами: за допомогою Інтернету, за допомогою підключення пристроїв до однієї мережі Wi-Fi, та за допомогою Bluetooth. Наразі додаток не дозволяє робити знімки екрану, але, як зазначають розробники – вихідний код застосунку повністю загальнодоступний для вивчення [3], тож це не заважає змінити логічну модель застосунку та вимкнути це блокування (рис. 1.1).

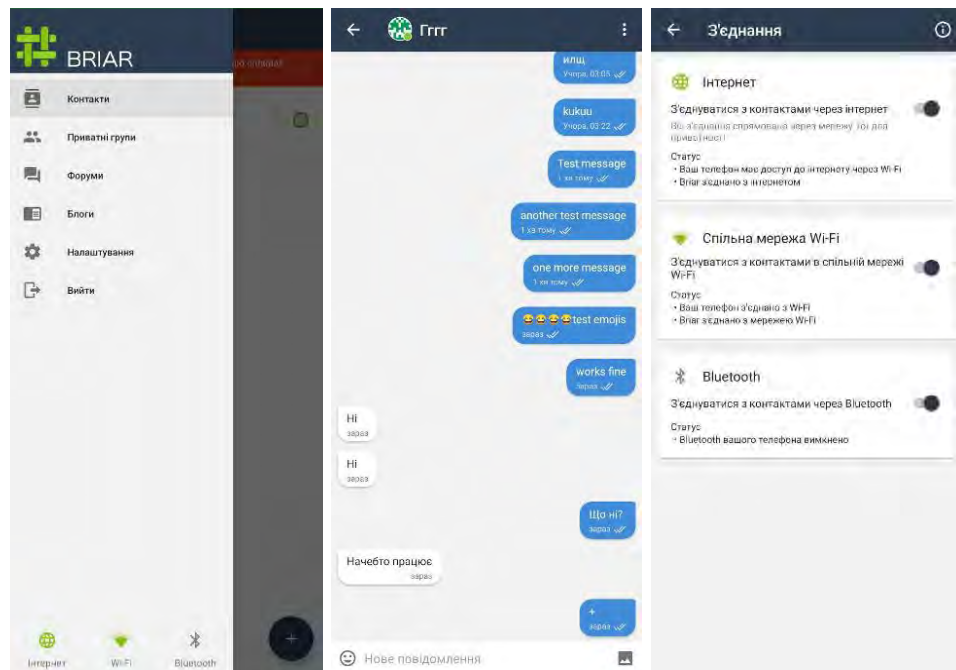


Рисунок 1.1 – Відображення застосунку «Briar»

1.2.2 Застосунок «Bridgefy»

Перш за все можна зазначити, що інтерфейс додатку «Bridgefy» [4] було вироблено у стилі Material Design, а гама складається із двох кольорів – червоного та білого. Плавність анімацій дає відчуття легкості додатку. Із недоліків – це набридлива реклама, що з'являється кожного разу входячи до форми чату (рис. 1.2).

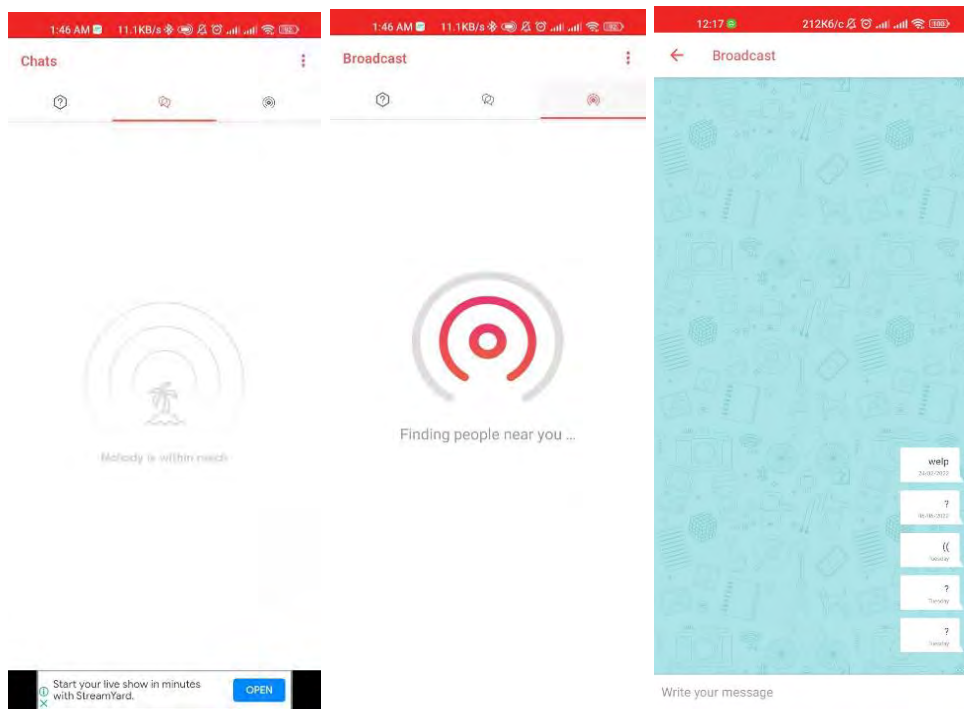


Рисунок 1.2 – Зображення інтерфейсу застосунку «Bridgefy»

1.2.3 Застосунок «Talkie»

Застосунок «Talkie» [5] не потребує налаштувань та дає змогу користувачу лише обрати ім'я. Після цього основний функціонал миттєво стає доступним користувачеві. Це є великим плюсом, бо, порівнюючи із Briar, не потрібно реєструватись, та проходити додаткові вікна. Функціонал додатку вражає. Є можливість не тільки переписуватись, а й обмінюватись медіа файлами, а також дзвонити через мережу Wi-Fi. До того ж наявний функціонал переписування у груповому чаті (рис. 1.3).

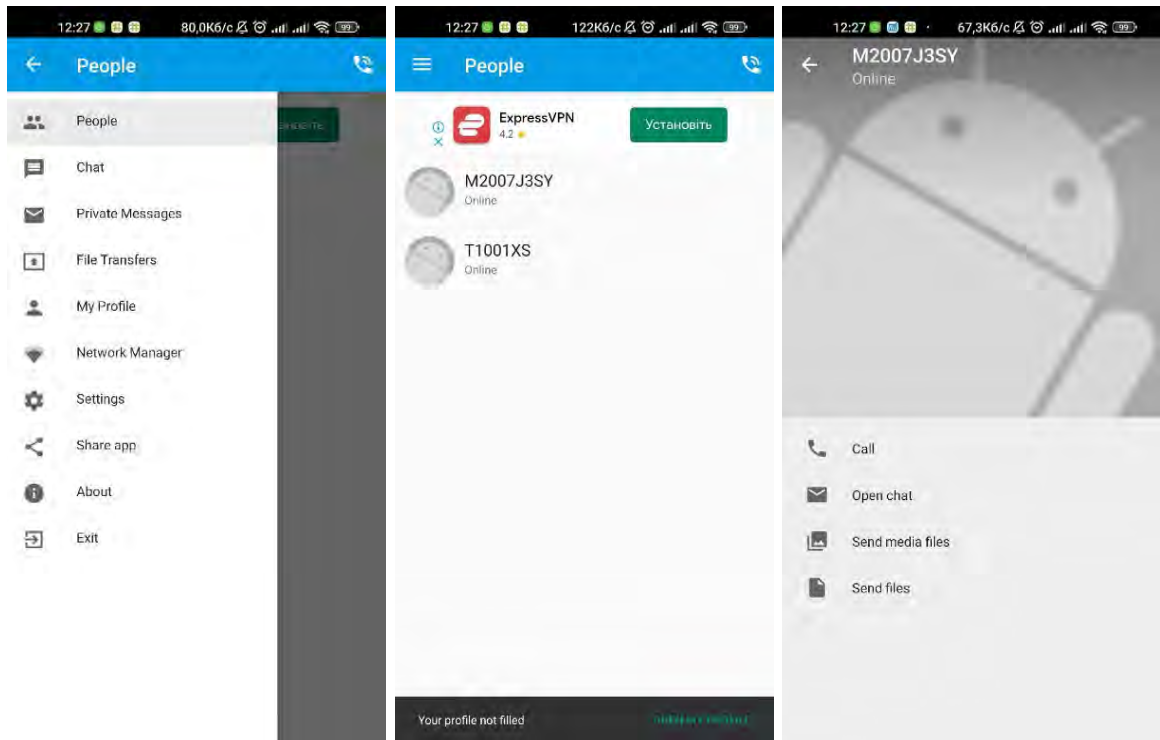


Рисунок 1.3 – Інтерфейс застосунку «Talkie»

1.2.4 Порівняльна таблиця програмних продуктів

Нижче в таблиці 1.1 відображено порівняльну характеристику аналогів та застосунком, що розроблюється. У якості оцінок виставляються мінуси та плюси. Чим більше плюсів отримано, тим краще себе показав застосунок під час порівняння за певним критерієм.

Таблиця 1.1 – Порівняльна характеристика застосунків-аналогів

Критерії оцінки	Назва застосунку			
	Bridgefy	Briar	Talkie	Juicy Wi-Fi Chat
1	2	3	4	5
Інтуїтивна зрозумілість	-	+	+++	+++
Зручність	±	+	++	+++
Сучасність дизайну	++	+	-	++

Продовження таблиці 1.1

1	2	3	4	5
Споживання ОЗУ	Від 70МБ	Від 60 до 100МБ	Від 90МБ	Від 40 до 100МБ
Споживання електроживлення	≈100 mAh	≈180 mAh	≈220 mAh	≈190 mAh
Стабільність роботи	-	++	+	++

Результати статистики споживання ресурсів отримано за годину активного користуванням додатку в однакових умовах.

Можна зазначити, що Bridgefy показав себе як додаток із приємним інтерфейсом, але при цьому жахливим функціоналом та загалом роботою. При тестуванні на інших пристроях Bridgefy викидував помилки та завершав роботу самостійно.

Briar показався стабільним застосунком, але є нюанс. Задля того щоб з'єднатись із пристроєм потрібно сканувати QR-код, або використати спеціально згенероване посилання, яке якось потрібно відправити співрозмовнику. Це, авжеж, вирішує основні проблеми зазначені в огляді предметної області, але такий варіант з'єднання не є зручним у деяких ситуаціях.

Одним із найкращих з функціоналу, але при цьому й найгіршим за критерієм «Сучасність дизайну» є Talkie. Його функціонал вражає: наявні групові чати, можливість відправлення медіа файлів, можливість редагування профілю, та, що не менш цікаве, підтримка дзвінків.

Juicy Wi-Fi Chat вирішує усі ці проблеми, що зустрічаються у аналогів та надає користувачу стабільний застосунок для спілкування.

1.2 Висновки

У розділі 1 було проведено аналіз предметної області розробки. Також було проведено детальний аналіз трьох найпопулярніших додатків, виявлення їх недоліків та переваг, та вирішення їх у застосунку що розроблюється. Таким чином розробку ПЗ можна вважати доцільною, адже вміщує у собі весь той функціонал, що потрібен саме для зручного спілкування у критичних ситуаціях.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ ЯКА ПРОЕКТУЄТЬСЯ

У розділі розглянуті основні особливості мов програмування, вибір інструментарію та огляд структури за стосунку.

2.1 Вибір інструментарію розробки програмного забезпечення

2.1.1 Вибір платформи та операційної системи, для якої розроблюється застосунок

Так, як ідея застосунку полягає в тому, щоб надати користувачу послуги для спілкування у критичних ситуаціях, то було розглянуто лише платформу мобільних пристроїв. У цю категорії також попадають планшети, адже вони, зазвичай, наділені тим самим функціоналом, що і мобільні пристрої. У якості операційної системи, під яку проектується застосунок, було обрано Android, адже вона є найбільш популярною, а отже й охопат користувачів має бути більше. Згідно за статистикою [6] Android домінує на світовому ринку, та відповідає за 71% всього використання смартфонів у першому кварталі 2021 року. За останні роки частка ринку Android падає через те, що iPhone 11 та iPhone 12 були добре сприйняті клієнтами (рис. 2.1).

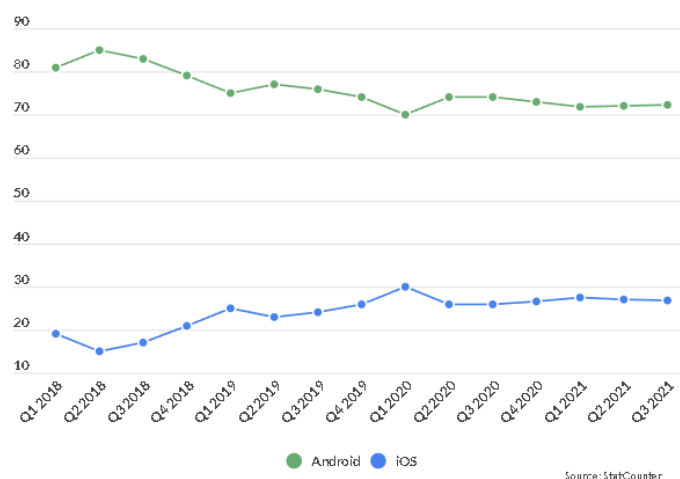


Рисунок 2.1 – Статистика глобального ринку Android та IOS [6]

Android вважається молодого операційною системою. Її перша версія була випущена 23 вересня 2008 року із назвою Astroboy. Цікавий факт, із розвитком та випуском нових версій, скорочене кодове ім'я версії становить літеру, що є назвою десерту [7].

Окрім цього, розвиток операційної системи Android не стоїть на місці. За останні роки було випущено безліч нових версій системи, кожна з яких має свої особливості та поліпшення у роботі. Так, наприклад, остання система, що була продемонстрована компанією Google – це Android 12 Snow Cone. Було створено новий підхід дизайну – «Material 3», або «Material You». Ідея дизайну – наділення елементів однією кольоровою гамою із системною, а елементи управління повинні бути збільшеними (рис. 2.2).

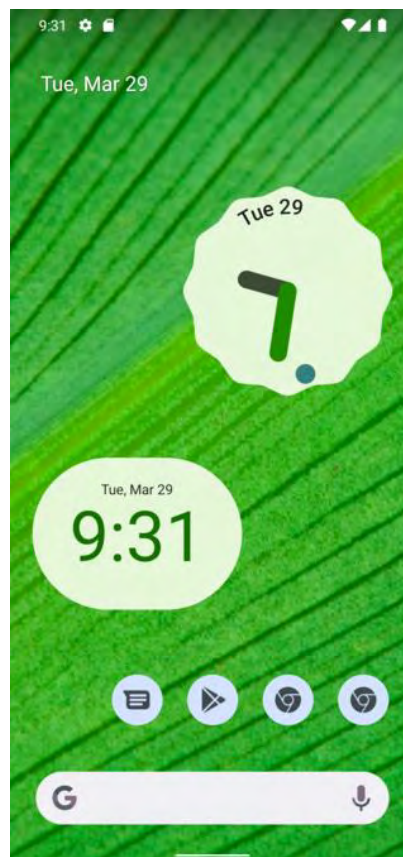


Рисунок 2.2 – Вигляд інтерфейсу Android із «Material You» дизайном [7]

У розробці під операційну систему Android є як переваги, так і недоліки. Серед переваг це багата технічна база та кількість документації, що докладно

описує той чи інший функціонал, або підхід до розробки компонентів. До того ж, на відміну від розробки під IOS, користувачу не обов'язково мати пристрій із системою під яку він розроблює. Займатись розробкою під IOS можуть тільки ті, хто має продукцію Apple та особистий акаунт у сервісах Apple.

Серед недоліків – це відносно часті оновлення систем, адже для підтримки як нових, так і старих пристроїв кодова база застосунку може значно збільшитись у розмірах, а підтримка такого коду стає великою проблемою як для розробника, так і для компанії. Для прикладу візьмемо останнє оновлення ОС Android. Серед змін вказано, що було втілено тотальний контроль над застосунками, що працюють у фоні. І якщо, наприклад, застосунок розроблюється для старих версій, то робота програми може кардинально відрізнятись, в залежності від версії системи.

2.1.2 Вибір середовища розробки

Для розробки застосунку під операційну систему Android створено офіційне програмне забезпечення – Android Studio. Воно було вперше представлено на конференції Google I/O 16 травня 2013 року, через 5 років після дати релізу першої версії системи Android. На поточний час розвиток Android Studio має шалений темп, адже продукт розроблюється із компанією JetBrains, що є славнозвісними на ринку інтегрованих середовищ, що вважаються одними з найкращіших продуктів для розробки та відлагодженню ПЗ. Починаючи з липня 2021 року команда розробників Android Studio почала випускати іменовані версії студії. За таким ж принципом як і версії Android випускаються версії Android Studio, за однією лише різницею. Замість кодових імен десертів використовується назви тварин. Перелік таких версій:

- Arctic Fox – дата релізу липень 2021 року;
- Bumblebee – дата релізу січень 2022 року;
- Chimpunk – дата релізу травень 2022 року.

Dolphin – доступна бета версія, стабільна версія ще не була представлена.

Основні особливості розробки у цьому середовищі є:

- підказки щодо оптимізації коду, запропонування рішень та застереження щодо використання тих чи інших компонентів у програмі для різних версій Android версій, що підтримуються додатком;
- наявний багатий набір шаблонів (layouts) для розробки форм (фрагментів або активіті);
- редагування макетів та перегляд результатів у режимі реального часу;
- наявність відлагодження та запуск застосунку через Wi-Fi ADB, що спростовує розробку;
- підтримка створення емуляторів системи Android всередині самої студії.

Для розробки Android застосунку існує ще середовище IntelliJ IDEA від команди Jet Brains. Проаналізувавши інтерфейс продукту, можна сказати, що вона має більше функціоналу порівнюючи з Android Studio. Є можливість роботи із базами даних, перегляд даних та робота з консоллю SQL (мова структурованих запитів, що використовується для роботи із базами даних). Окрім цього немало важливою особливістю для розробки та аналітики розроблених даних є підтримка відображення діаграм, що зображує зв'язки наслідувань, перелік полів та методів, що можна фільтрувати за модифікаторами доступу. Нажаль, такого функціоналу немає в Android Studio, та досягається завдяки встановленню плагінів у IDE, наприклад Plantuml та PlantumlParser. Але є нюанси, вони розглянуті у п. 2.1.4.

Ці два середовища розробки мають однаковий функціонал із роботою з системою контролю версій, перегляду та редагування модулів проєкту, аналіз та смарт рефакторинг коду.

2.1.3 Вибір мови програмування для розробки мобільного додатку

Мови програмування – то є лише інструмент, що описує алгоритми розроблені користувачем. Але, тим не менш, він є важливою частиною у створенні продукту, адже потрібно чітко проаналізувати наперед можливість підтримки кодової бази, наявність потрібних та актуальних бібліотек, що повинні бути використані у проекті. На сьогодні розробка Android застосунку можлива на самих різних мовах: C#, C++, скриптові мови програмування Python та JavaScript. Але є дві основні, що вважаються офіційними або базовими для розробки продуктів: Java та Kotlin.

Java є офіційною мовою розробки під систему Android та підтримується середовищем розробки Android Studio. Вона вважається основою, що потрібно вивчити задля ефективного програмування застосунків обраної системи. Але часи змінюються і на заміну приходить нова, покращена мова програмування Kotlin [8].

Kotlin, слід зазначити, є обгорткою для мови Java, бо код написаний цією мовою спершу компілюється у java-код, а вже потім усі вихідні дані передаються до java компілятора [9] (рис. 2.3). Вона була створена компанією Jet Brains задля лаконічного та типо-безпечнішого написання коду ніж Java. Вона розроблюється з 2010 року, а публічно представлена була у липні 2011р.

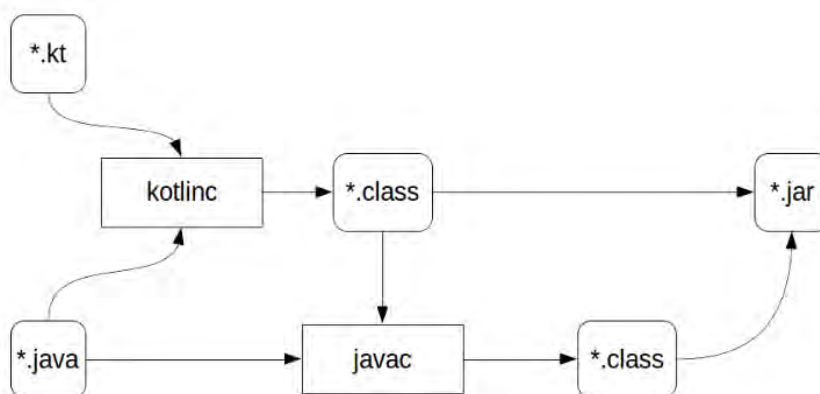


Рисунок 2.3 – Зображення послідовності компілювання коду Kotlin та Java [9]

Особливостями мови коди є лаконічний вигляд, надає корутини (співпрограми, що дозволяють працювати асинхронно). Ще однією важливішою особливістю мови є функціонал розширення(extensions). Це дозволяє розширювати функціонал класу шляхом додавання нового функціоналу без втручання або наслідування цього класу. Приклад функції-розширення.

```
fun View.isVisibleOrGone(isVisible: Boolean) {
    if (isVisible) {
        this.makeVisible()
    } else {
        this.makeGone()
    }
}

fun View.makeInvisible() { this.visibility = View.INVISIBLE }
fun View.makeVisible() { this.visibility = View.VISIBLE }
fun View.makeGone() { this.visibility = View.GONE }
```

Як можна побачити із коду вище створення такої функції відбувається завдяки прописуванню класу-контексту (у нашому разі це View), та імені функції, що розділена крапкою. У тілі такої функції будуть доступні усі public поля та методи класу-контексту. На прикладі розглянуто зміна видимості візуального об'єкта класу View завдяки зміні поля View::visibility.

Це значно спрощує розробку, адже допомагає виділяти код та виносити до іншого модулю, аби уникнути повторень.

2.1.4 Висновки за розділом 2.1

Проаналізувавши інформацію про середовища розробки та особливостей базових мов програмування для розробки під систему Android було вирішено використовувати Android Studio, як основну програму для розробки.

Також було вирішено використовувати IntelliJ IDEA як додаткову, задля перегляду та створення діаграм класів що розроблюються. Причиною цього стала відсутність підтримки аналізу Kotlin файлів плагіном Plantuml.

2.2 Структура файлів Android за стосунку

Кожний застосунок, що розроблюється під операційну систему Android має чітко визначену структуру розположення файлів. У корні проекту знаходяться файли Gradle, що описують процес збірки додатку у бінарний файл для подальшого встановлення у систему Android, та модулі проекту. При створенні нового проекту студія генерує головний модуль із назвою «app» («application»), що є точкою входу у застосунок. Доступно чимало типів модулів для створення при розробці Android застосунку. Перелік таких модулів наведено на рисунку 2.4.

Модуль – це контейнер, що зберігає вихідний код програми, файли ресурсів та файлів конфігурацій для збірки. Кожний модуль може компілюватись незалежно від інших, тож на великих проектах такий підхід, як багатомодульність, полегшує роботу розробникам та покращує читаємість коду.

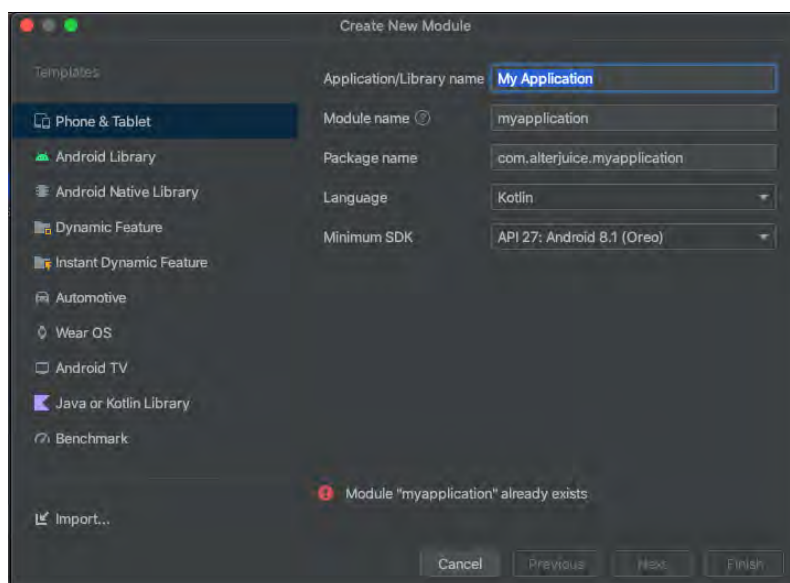


Рисунок 2.4 – Перелік шаблонів-модулів, що доступні для вибору

Для розробки чат-додатку потребуються модулі «Phone & Tablet», «Android Library» та «Java or Kotlin Library». Інші модулі допомагають розробити програму під інші системи, бо мають відповідні бібліотеки для роботи зі смарт-годинниками або телевізорами, та автомобільними пристроями [10].

Модулі «Phone & Tablet» та «Android Library» схожі між собою вмістом файлів маніфесту та ресурсів, але між ними є різниця. На відміну від «Phone & Tablet», що компілюється у файл APK, «Android Library» компілюється у файл Android Archive (AAR). Такий підхід дозволяє розроблювати функціонал та підключати цей модуль як залежність у інші проекти [11].

Модуль «Java or Kotlin Library» дозволяє вміщувати у собі лише код Java або Kotlin файлів. Він не має маніфесту та файлів ресурсів. Мета цього типу модуля – відокремити бізнес-логіку, розділити на абстракції та реалізації, та надати доступ через інтерфейси використовуючи відповідні модификатори доступу зазначених у мовах програмування Java та Kotlin.

Розроблюючи проект можна побачити такі дерикторії:

- Manifest – директорія, що зберігає файл AndroidManifest.xml який є описом програми, перелік усіх компонентів Android: Activity, Service, Receiver, ContentProvider;
- Java – директорія що містить кодову базу;
- Res – директорія, в якій зберігаються усі файли ресурсів, що можуть бути використані для роботи застосунку.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

В даному розділі розглянуто архітектуру розробленого застосунку, використані патерни проектування, ознайомлено з абстракції та їх реалізаціями, алгоритми та рішення щодо інтерфейсу.

3.1 Реалізація архітектури та патернів програмування

3.1.1 MVVM як основний архітектурний підхід у розроблюваного продукту

До вибору архітектури застосунку потрібно підходити відповідально, адже на початковому етапі це сприяє зручному відлагодженню, читаємості та повторному використанню коду. А у перспективі розробки – запобігає появленню труднощів із підтримкою проєкту.

Серед усіх популярних архітектур була обрана MVVM (Model-View-ViewModel). Вона вирішує проблему MVP – перехресних посилань між презентером та відображенням(View). Окрім цього це дає змогу відділити реалізацію бізнес-логіки від логіки роботи інтерфейсу. Таким чином ViewModel нічого не знає про View з яким працює, а оновлення даних відбувається за допомогою реактивного підходу - використання поведінкового шаблону «Спостерігач».

Сумуючи все вищезазначене – ми отримаємо відмінний підхід до розробки проєкту. Відокремлення бізнес-логіки від відображення дає змогу розділяти реалізації від абстракцій, а це у свою чергу підштовхує імплементувати багатомодульний підхід, що насамперед прискорює процес збірки, тестування та загалом розробки додатку як одному, так і у команді, розподіляючи роботу виключно у межах модулю кожному розробнику.

3.1.2 Патерни проектування, що реалізовані у за стосунку

«Спостерігач» – це патерн проектування, який створює механізм підписки, та дає змогу одним об'єктам стежити й реагувати на події, які відбуваються в інших об'єктах [12]. Оскільки робота додатку передбачає постійного підключення до мережі Wi-Fi, то було вирішено створити об'єкт, що має інформацію щодо статусу підключення. Статус підключення працює згідно з патерном «Спостерігач», щоб вчасно інформувати систему та підписників щодо нових змін, а також повторює патерн об'єкт-одинак або, як його ще називають, «Singleton».

Ідея «Singleton» полягає у гарантії створення та зберігання лише одного екземпляру класу у момент роботи програми, що може мати глобальну точку доступу до нього [13]. Прикладом такого патерну є створення єдиного екземпляру бази даних, або зберігання констант. Це зроблено задля того щоб бути впевненим, що дані будуть актуальні в усіх частинах роботи додатку. Об'єкт-одинак імплементовано у мову Kotlin на рівні із іншими типами класів і його можна створити за допомогою ключового слова «object {ім'я}». Зазвичай такого підходу достатньо для зберігання даних під час роботи програми, але якщо є потреба зробити доступ до екземпляру потокобезпечним, то потрібно реалізувати це самостійно [14]. Реалізація сінглтону із потокобезпечним звертанням є у класі DataRepo та у класах, що наслідуються від користувацького класу Storage. Додатковою інформацією щодо реалізації статусу підключення є те, що він також повторює поведінковий патерн «Стан» [15], ідея якого є мати заздалегідь визначений перелік станів, що можуть без проблем знати один про одного та мати інтерфейс переходу від одного стану до іншого. На рисунку 3.1 зображено реалізація класу ConnectivityStatus, що описує стан підключення до серверу та має такий перелік:

- NotConnected – не підключений до серверу;

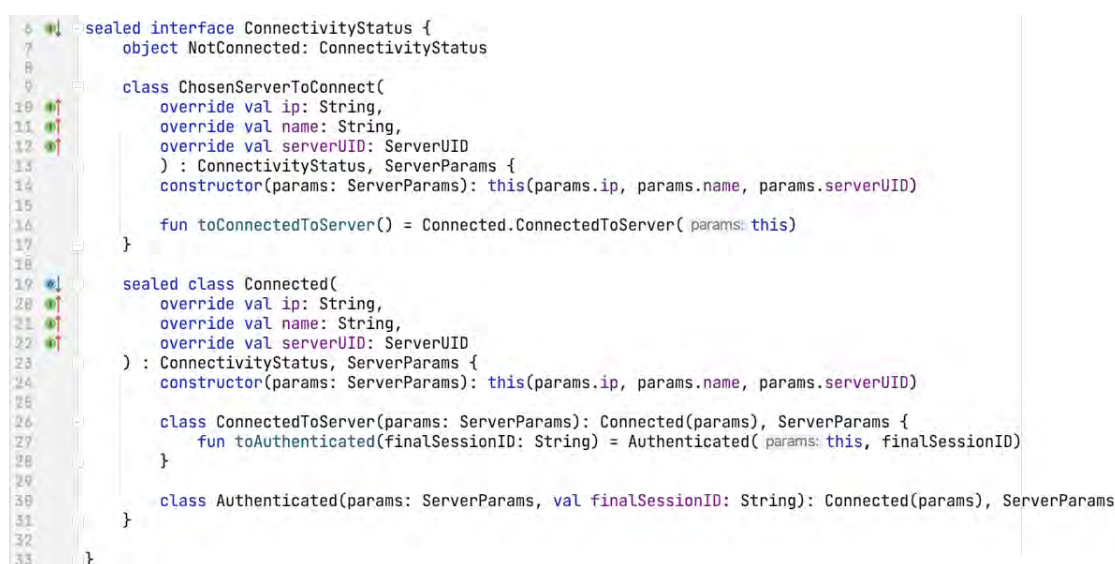
– ChosenServerToConnect – стан, що зберігає інформацію про обраний користувачем сервер, до якого потрібно підключитись. Має поля IP серверу, назву серверу та його ідентифікаційний номер.

Стан «Connected» має ще два додаткових стану, що логічно різняться між собою:

– ConnectedToServer – стан, що інформує систему про успішне з'єднання з віддаленим сервером;

– Authenticated – стан, на який змінюється ConnectedToServer у разі успішного підключення та авторизації на віддаленому сервері іншого або цього пристрою.

Усі вони наслідуються від класу-контексту ConnectivityStatus.



```

sealed interface ConnectivityStatus {
    object NotConnected: ConnectivityStatus

    class ChosenServerToConnect(
        override val ip: String,
        override val name: String,
        override val serverUID: ServerUID
    ) : ConnectivityStatus, ServerParams {
        constructor(params: ServerParams): this(params.ip, params.name, params.serverUID)

        fun toConnectedToServer() = Connected.ConnectedToServer( params: this)
    }

    sealed class Connected(
        override val ip: String,
        override val name: String,
        override val serverUID: ServerUID
    ) : ConnectivityStatus, ServerParams {
        constructor(params: ServerParams): this(params.ip, params.name, params.serverUID)

        class ConnectedToServer(params: ServerParams): Connected(params), ServerParams {
            fun toAuthenticated(finalSessionID: String) = Authenticated( params: this, finalSessionID)
        }

        class Authenticated(params: ServerParams, val finalSessionID: String): Connected(params), ServerParams
    }
}

```

Рисунок 3.1 – Вигляд класу ConnectivityStatus, що повторює патерн «Стан»

Було також застосовано патерн «Адаптер», який дає змогу об'єктам із несумісними інтерфейсами працювати разом. Найпопулярніше використання – це реалізація адаптеру, що приймає користувацькі дані та перетворює їх згідно за шаблоном у інтерфейсі додатку. У Андроїд поширеними адаптерами є ListAdapter та RecyclerView.Adapter.

3.1.3 Вибір інструменту розпаралелювання задач

Основою будь-якого застосунку є його головний потік. В Android системі це не виключення, тож вона запускає усі додатки на головному(Main) потоці. За замовчуванням система Android має такі потоки: Default, IO, Main. Main потік ще називається UI-потік, і вважається головним, бо з нього створюються інші потоки та оновлюється інтерфейс користувача. Однак для роботи важких або довготривалих задач маємо створювати додаткові або використовувати існуючі.

Важкими або довготривалими задачами є зчитування файлів, відправлення даних та прослуховування віддалених серверів. Для вирішення декількох задач одночасно якраз використовують розпаралелювання. Для створення потоків доступний стандартний функціонал класу Thread, що знаходиться у бібліотеці Java (пакет `java.util.concurrent`). Він створюється за допомогою переданого у конструктор об'єкту `Runnable`, що зберігає опис роботи, що буде виконуватись. Після цього достатньо викликати метод `start()` у створеного об'єкту `Thread`. Для зупинки потоку існує метод `stop()`, але його використання не рекомендується, так як потік не зможе вивільнити зайняті їм ресурси. Краще використовувати метод `interrupt()`, а довготривалі операції обгорнути у блоки `try/catch`. Так як потоки працюють ізольовано, то для отримання якогось результату використовують реактивний підхід або функції зворотного виклику (`callback`).

Ідея реактивного програмування полягає в реалізації патерну «Спостерігач» (див. розділ. 3.1.1), де наявний об'єкт даних і підписники, що реагують на його зміни. Приклад таких даних є `LiveData/MutableLiveData`. Вони дають змогу підписуватись на оновлення даних та зв'язуватись із життєвим циклом форми в Android. Підписуватись на дані можна за допомогою двох методів: `observe` та `observeForever`. Перший спосіб – це використання `observe`, що очікує два аргументи – `ViewLifecycleOwner` та функцію зворотного виклику, що буде викликатись на зміни у даних. `ViewLifecycleOwner`

використовується для того, щоб зупинити підписника отримувати дані поки це не має необхідності (у разі якщо програма була перекрита іншим додатком).

Асинхронність та багатопотоковість – це різні речі. Виділення потоку – крайньо важка операція, тож не потрібно зловживати цим. При асинхронному програмуванні задачі будуть ділитись на одному потоці та поділяти процесорний час. Якщо такої можливості немає, то створюється додатковий потік.

RxJava та Kotlin Coroutines (далі – корутини) схожі між собою. Вони вміщують механізми асинхронності та патерн «Спостерігач». Також вони мають приблизно один функціонал, а саме: можливість створення даних, підписки на дані, наявність операторів трансформування даних(`map`, `flatMap`, `transform`, `filter`), та вибір потоку, на якому ці дані будуть створюватись, змінюватись та отримуватись підписниками. Це реалізується за допомогою методу `flowOn(..)` та аргументу одного із значень з класу `Dispatchers`. Єдина різниця це те, що Kotlin Coroutines є нативним механізмом мови програмування Kotlin. Визначення та перехід від одного потоку на інший виконується за допомогою вищевказаного класу `Dispatchers`, що має заздалегідь визначені поля що описують потоки: `Main`, `Default` та `IO`. Для коректної роботи з корутинами потрібен `CoroutineScope`. За замовчуванням існує `GlobalScope`, а у додаткових бібліотеках використовується `LifecycleScope`, що залежить від життєвого циклу форм активіті та фрагментів, та `ViewModelScope`, що використовується у `ViewModel` компонентах та відповідає за життєвий цикл вьюмоделі.

Аналогом до `LiveData` та `MutableLiveData` у Kotlin Coroutines є клас `StateFlow` та `MutableStateFlow`. Різниця між `StateFlow` та у `MutableStateFlow` полягає в тому, що перший є інтерфейсом для другого, тож він має функціонал підписки на дані, а вже `MutableStateFlow` дозволяє оновлювати дані за допомогою методу `setValue(..)`. Такий підхід дозволяє одним об'єктам лише зчитувати дані, а іншим як зчитувати, так і оновлювати.

Приклад підписки на оновлення даних використовуючи Kotlin Coroutines:

```

// Створення об'єкту StateFlow із початковим значенням
val someStateFlow: StateFlow<String> = MutableStateFlow("Some initial
value")

...
lifecycleScope.launch(Dispatchers.IO) {
    // підпис та збір даних відбувається на потоці IO
    stateFlow.collect { newValue ->
        withContext(Dispatchers.Main){ // Перехід на головний потік
            // Робота з даними відбувається на головному потоці
            Toast.make(this, "Value updated: $newValue",
Toast.LENGTH_SHORT).show()
        }
    }
}

```

Під час розробки було вирішено використовувати Kotlin Coroutines, бо насамперед це є нативним механізмом мови, що була використана для розробки. Технологія стає більш популярною порівнюючи із RxJava. Окрім цього робота з корутинами набагато зручніша та дає коду більшу читабельність.

3.1.4 Спосіб збереження даних

Для збереження користувацьких даних було обрано два формати: JSON та XML.

Для роботи з JSON даними було обрано бібліотеку Gson. Вона розроблена командою Google та полегшує кодування та декодування даних у структурований вигляд. Для прикладу візьмемо клас «ConnectivityStatus», що був розглянутий у розділі 3.1.2. Так як клас має декілька станів, а стани мають різні параметри, що потрібно зберігати – на допомогу прийде розроблений командою Google адаптер – RuntimeTypeAdapter. Для його роботи потрібно визначити тип, що буде зберігатись, та зареєструвати його підтипи перед створенням об'єкту Gson через GsonBuilder().

Виглядає це наступним чином.

```

val adapter = RuntimeTypeAdapter.of(ConnectivityStatus::class.java).apply {
    // реєстрація підтипів класу-контексту
    registerSubtype(ConnectivityStatus.NotConnected::class.java)
    registerSubtype(ConnectivityStatus.ChosenServerToConnect::class.java)
    registerSubtype(ConnectivityStatus.Connected.ConnectedToServer::class.java)
    registerSubtype(ConnectivityStatus.Connected.Authenticated::class.java)
}

```

Будуємо об'єкт gson.

```

val gson = GsonBuilder()
    .registerTypeAdapterFactory(adapter)
    .setPrettyPrinting()
    .create()

```

Після цього ми можемо кодувати та декодувати дані за допомогою методів `toJson(..)` та `fromJson(..)`. Приклад кодування даних `ConnectivityStatus` класу в залежності від переданого стану (рис. 3.2).

```

[NotConnected]
{
  "status": {
    "type": "NotConnected"
  }
}

[ChosenServerToConnect]
{
  "status": {
    "type": "ChosenServerToConnect",
    "ip": "192.168.0.176",
    "name": "ServerName",
    "serverUID": "d42aa343-9b63-4fc0-af65-e21ef876a402"
  }
}

[ConnectedToServer]
{
  "status": {
    "type": "ConnectedToServer",
    "ip": "192.168.0.176",
    "name": "ServerName",
    "serverUID": "d42aa343-9b63-4fc0-af65-e21ef876a402"
  }
}

[Authenticated]
{
  "status": {
    "type": "Authenticated",
    "finalSessionID": "{someFinalSessionUUID}",
    "ip": "192.168.0.176",
    "name": "ServerName",
    "serverUID": "d42aa343-9b63-4fc0-af65-e21ef876a402"
  }
}

```

Рисунок 3.2 – Приклад кодування даних `ConnectivityStatus` класу за допомогою Gson бібліотеки та `RuntimeTypeAdapter`

Для збереження XML даних використовується клас `SharedPreferences`, що вбудований у `Android NDK`. Це дозволяє створювати сховище даних у вигляді ключ-значення. У поточної версії додатку зберігаються ім'я користувача, назва серверу, дані про сесії (перелік зареєстрованих користувачів) та закодовані у JSON об'єкти.

Окрім цього для збереження констант було використано об'єкти `Singleton`. Вони зберігають перелік каналів повідомлень, ключи для аргументів та текстові дані для повторного використання у додатку.

3.1.5 Навігація між вікнами додатку

Для реалізації навігації було обрано бібліотеку `NavigationUI Component` [16]. Вона з легкістю розширює існуючий функціонал переходів між вікнами активіті та фрагментами, дозволяє створювати та налагоджувати анімацію переходів. Також є підтримка передачі даних між вікнами завдяки заздалегідь визначеному списку аргументів, що притаманний тому чи іншому вікну.

Для початку роботи із `NavigationUI` потрібно створити файл навігації. В ньому й будуть визначені усі доступні вікна, зв'язки-переходи та список аргументів. На рисунку 3.3 відображено файл навігації, що використовується у проекті.

`Android Studio` дозволяє редагувати та налагоджувати усі зв'язки та конфігурації шляхів (рис. 3.4).

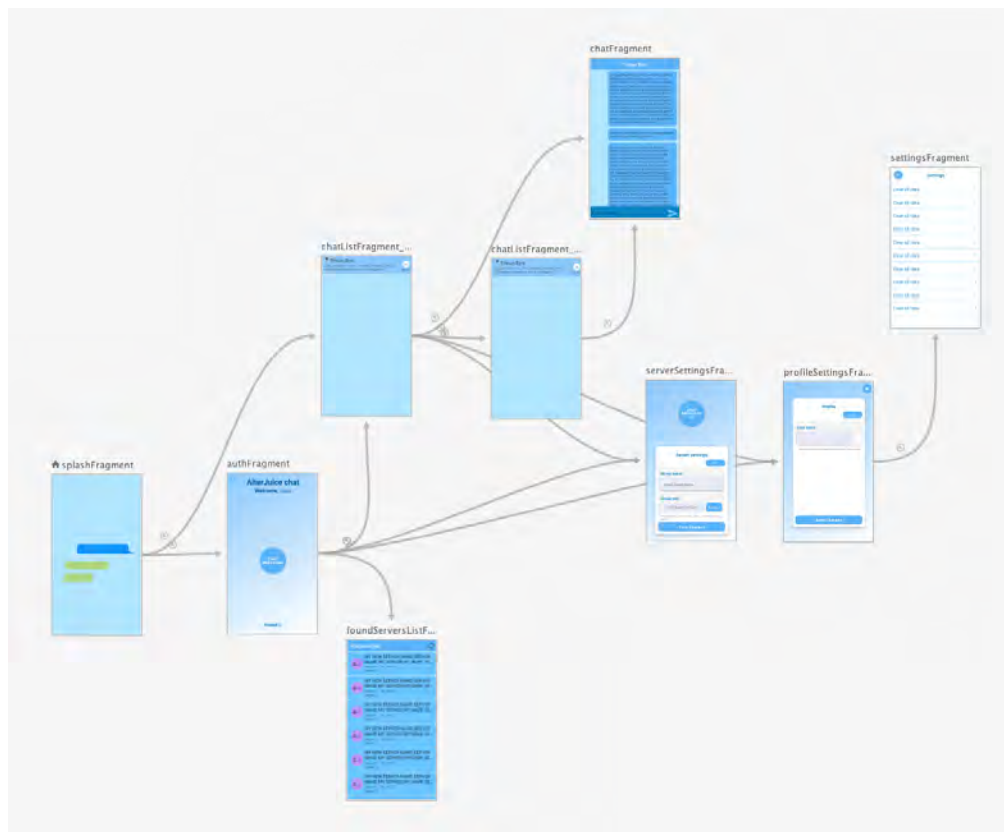


Рисунок 3.3 – Графічний вигляд зв'язків між вікнами, що реалізован за допомогою NavigationUI

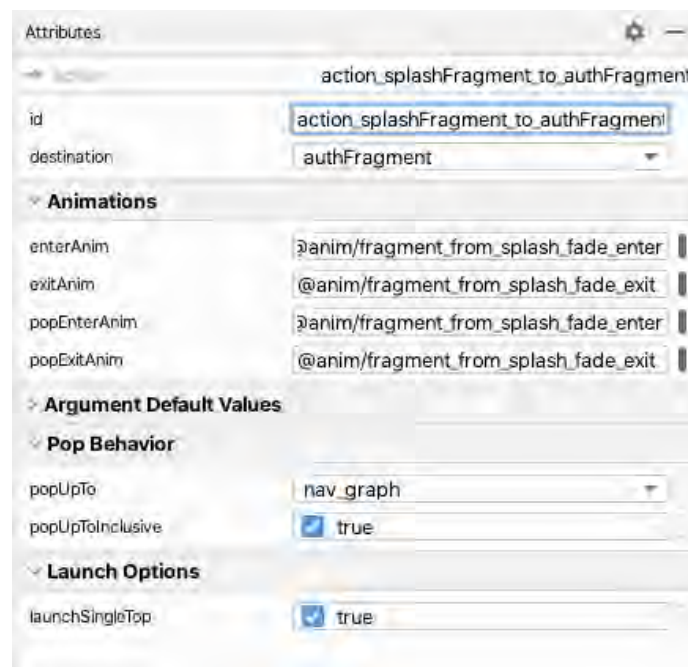


Рисунок 3.4 – Приклад конфігурації шляху-зв'язку, що поєднує екран завантаження та головний екран додатку

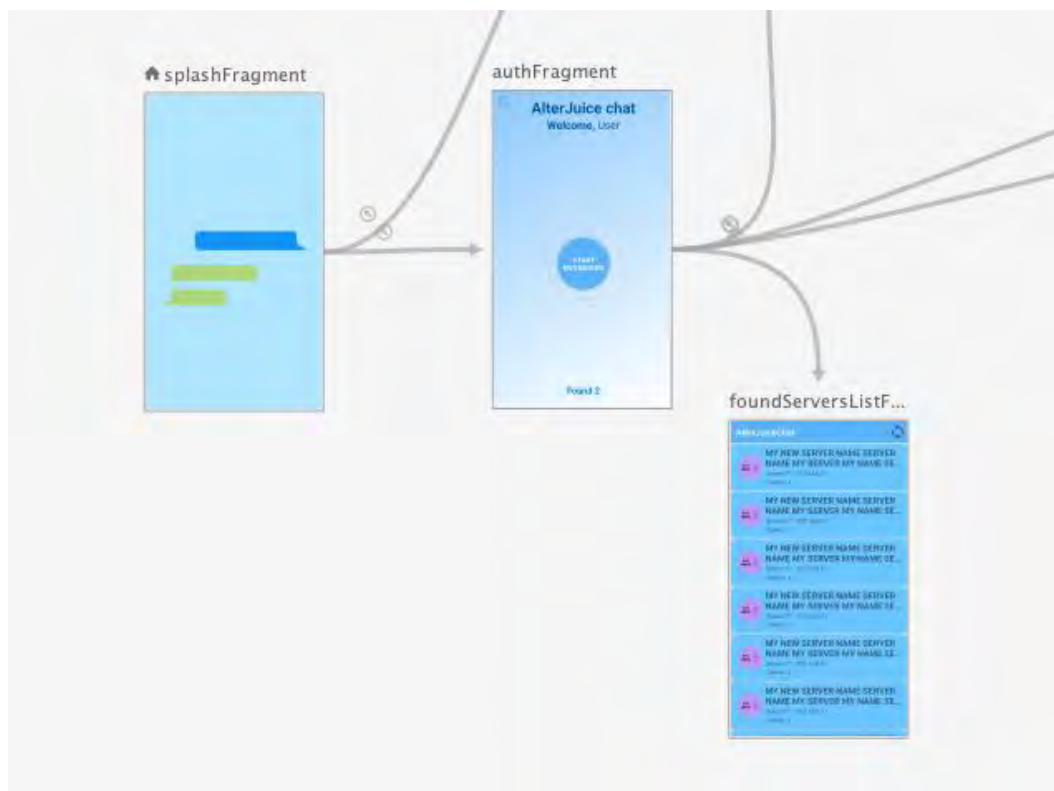


Рисунок 3.5 – Перегляд зв'язку між головним екраном та екраном, що відображає знайдені кімнати

Як можна побачити з рисунку 3.5 – було передбачено перехід до екрану зі списком доступних кімнат лише з головного екрану.

Перехід між екранами програмно здійснюється завдяки методу `navigate(..)`, що приймає один обов'язковий аргумент – шлях до вікна, до якого потрібно здійснити перехід. Окрім цього можна додати аргументи залучені у об'єкт `Bundle`, та конфігурації переходу – об'єкт класу `NavOptions`.

Для генерації дії-переходу до вікна, що далі буде використовуватись при натисканні на повідомлення використовуються допоміжні статичні функції визначені у коді додатку.

```
fun <TActivity: Activity> generatePendingIntentForNavigation(
    context: Context,
    activityClass: Class<TActivity>,
    @NavigationRes navigationGraphID: Int,
    @IdRes destinationResIDs: List<Int>,
    args: Bundle = bundleOf()
```

```

): PendingIntent {
/*
 * How to open fragment page, when pressed a notification in android
 * https://stackoverflow.com/a/55245317 */
return NavDeepLinkBuilder(context)
    .setComponentName(activityClass)
    .setGraph(navigationGraphID)
    .apply {
        destinationResIDs.forEach(::addDestination)
    }
    .setArguments(args)
    .createPendingIntent()
}

```

Приклад використання спільного коду, визначеного вище, для генерації дії, що буде виконана після кліку саме на текстове повідомлення.

```

fun generatePendingIntentForDirectChatDestination(context: Context,
userID: UserID, userName: UserDisplayName): PendingIntent {
    return generatePendingIntentForNavigation(
        context = context,
        activityClass = MainActivity::class.java,
        navigationGraphID = R.navigation.nav_graph,
        destinationResIDs = listOf(R.id.chatListFragment_NO_FILTER,
R.id.chatFragment),
        args = bundleOf(
            NavArgKeys.CHAT_USER_ID to userID,
            NavArgKeys.CHAT_USER_NAME to userName,
        )
    )
}

```

Метод `generatePendingIntentForNavigation` приймає об'єкт класу `Context`, клас, що являється підтипом класу `Активіті`, посилання на файл навігації, що зберігає шукані шляхи та список цих самих шляхів, що будуть додані у навігаційний стек. Це потрібно для того, щоб користувач, натиснувши на повідомлення, а потім на кнопку «назад» не вийшов із додатку, а повернувся,

наприклад, до вікна із переліком доступних діалогів. Саме такий перелік і зазначено у методі `generatePendingIntentForDirectChatDestination` у аргументі із назвою `destinationResIDs`.

3.1.6 Висновки за розділом 3.1

У розділі 3.1 було розглянуто основні архітектурні рішення та підходи щодо реалізації, внутрішньої роботи додатку та збереження даних. Також було розглянуто спосіб навігації між вікнами із використанням сучасних бібліотек. Наведено приклади, рисунки та схеми.

3.2 Огляд компонентів та розроблених класів

3.2.1 Огляд створених файлів та структури застосунку що розроблюється

Під час розробки застосунку було використано модульний підхід до компонування файлів. Було створено 6 модулів (рис. 3.6): один модуль «Phone & Tablet» із назвою `app`, та 5 додаткових модулів типу «Java & Kotlin Library».

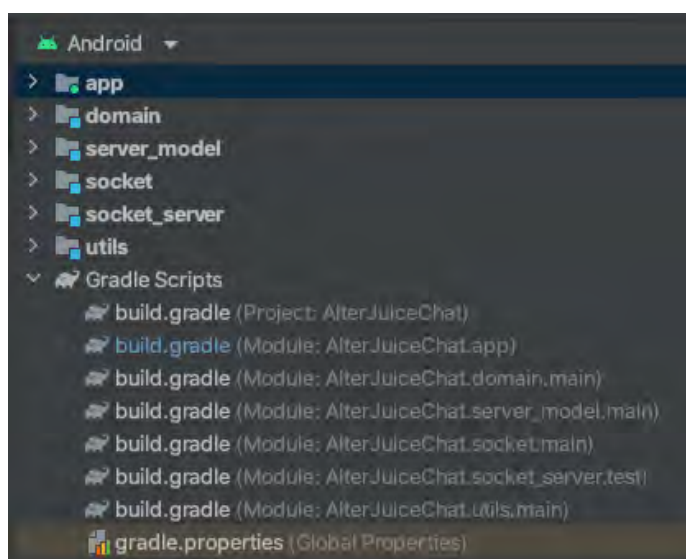


Рисунок 3.6 – Загальний вигляд модулів за стосунку

Опис модулів:

- App – зберігає усю кодову базу, що має відношення до Android компонентів. Модуль має пакети (packages), що логічно розділені на функціонал що зберігають, та мають відповідні назви (рис. 3.7);
- Domain – модуль зберігає деякі моделі даних, інтерфейси роботи із даними та файли із константними значеннями (рис. 3.8);
- Server_model – модуль, що зберігає модель об'єктів, що використовуються сервером під час обміну даними між пристроями (рис. 3.9);
- Socket – модуль, що зберігає клас, функціонал якого передбачає пошук мережевих кімнат (серверів) поблизу (рис. 3.10);
- Socket_server – модуль, що зберігає класи для коректної роботи серверу та коректного підключення до серверів (рис. 3.11);
- Utils – модуль, що зберігає допоміжні класи-утиліти (рис. 3.12).

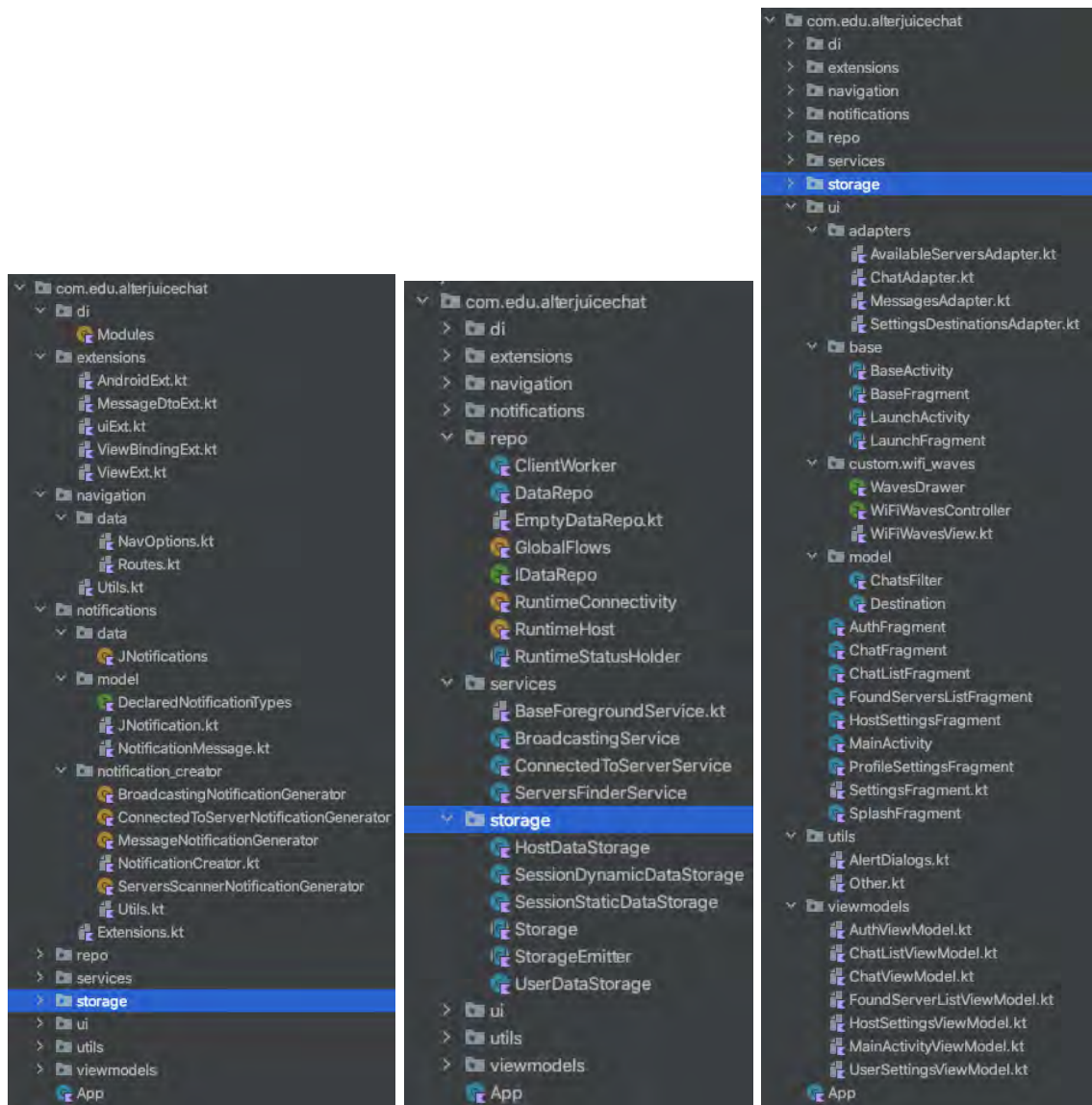


Рисунок 3.7 – Зображення структури модулю App

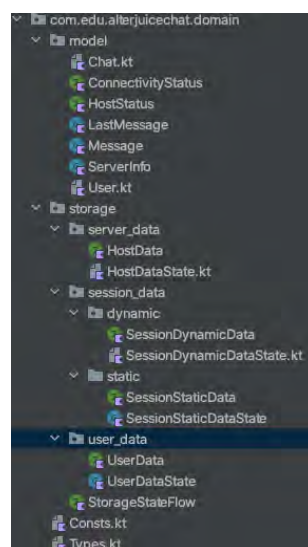


Рисунок 3.8 – Структура файлів модулю Domain

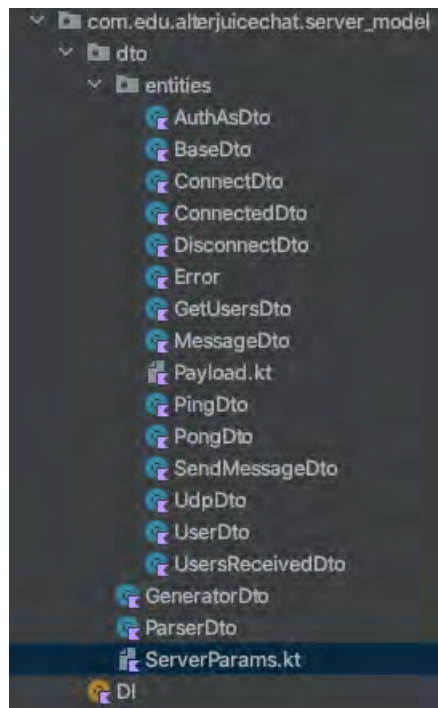


Рисунок 3.9 – Структура файлів модулю server_model

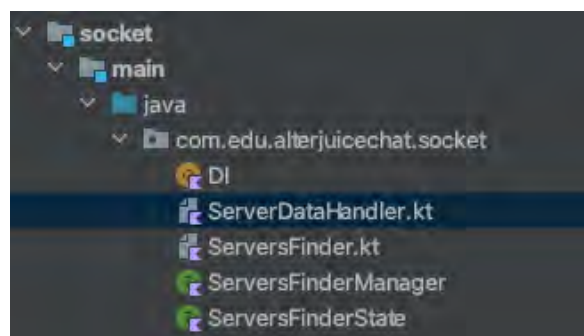


Рисунок 3.10 – Структура файлів модулю socket

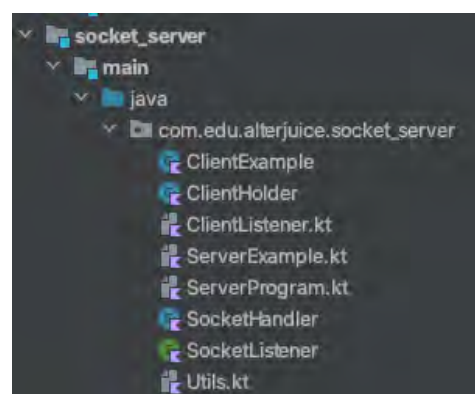


Рисунок 3.11 – Структура файлів модулю socket_server

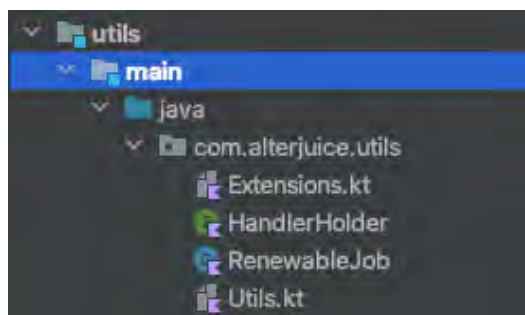


Рисунок 3.12 – Структура файлів модулю utils

3.2.2 Класи, що відповідають за відображення інтерфейсу

За візуальне відображення у системі Android відповідають класи фрагментів, активіті та адаптери, що перетворюють користувацькі типи даних у візуальний вигляд. Ці класи використовують файли макетів та зображень (layouts та drawables).

Файли макетів та іконок зберігаються у спеціальній директорії під назвою res/layouts та res/drawables.

3.2.2.1 Класи активіті. MainActivity

Застосунок використовує лише один фінальний клас активіті із назвою MainActivity. Окрім цього було розроблено базові класи: BaseActivity та LaunchActivity. Так як MainActivity це єдиний клас активіті, то він наслідує ці два абстрактних класи. Зображення зв'язку наслідування показано на рисунку 3.13.

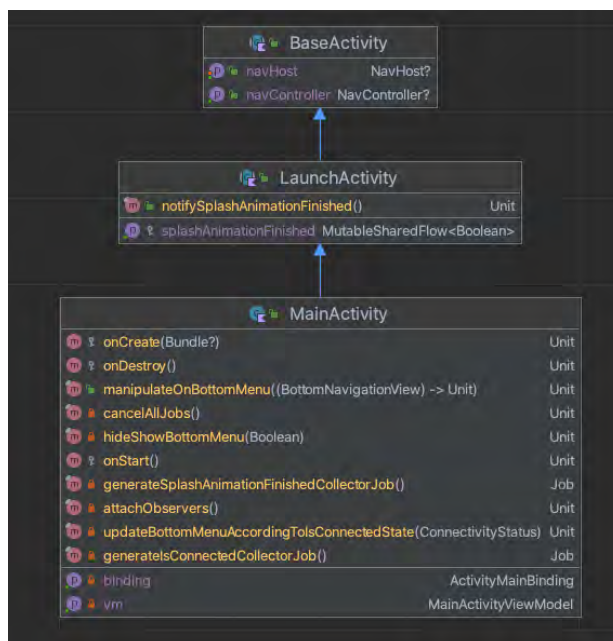


Рисунок 3.13 – Відображення діаграми, що демонструє наслідування класів-активіті від базового

Клас MainActivity є контейнером для фрагментів, та регулює роботу вікон, що повинні показуватись користувачу. Клас має поле типу MainActivityViewModel, та зберігає змінну, значення якої є поточний статус з'єднання із сервером. Під час оновлення значення викликається функція зворотного виклику. Далі відбувається перевірка на конкретний тип статусу, і вже на основі цього показується відповідні вікна, або зміна елементів меню.

3.2.2.2 Класи фрагментів та їх зв'язки

Фрагменти у системі Android використовуються задля відділення інтерфейсу на окремі вікна, що виконують зазначену роботу. Вони значно легші за класи-активіті та можуть бути повторно використані на відміну від активіті, що являються повноцінним вікном. Також, одночасно активіті може відображати декілька фрагментів водночас, вони можуть бути легко додані до вікна та прибрані. Цим займається FragmentManager клас за допомогою команди у контексті класу активіті.

```
supportFragmentManager
    .beginTransaction()
    .replace(R.id.mainFragmentContainer, SomeFragment())
    .commit()
```

Де `R.id.mainFragmentContainer` є ідентифікатором контейнера для фрагментів.

Було розроблено два базових класів фрагменту та 8 фінальних класів-фрагментів, кожний із яких розглянуто у п. 3.3. На рисунку 3.14 відображено зв'язки-залежності фрагментів від базового класу – `BaseFragment`.

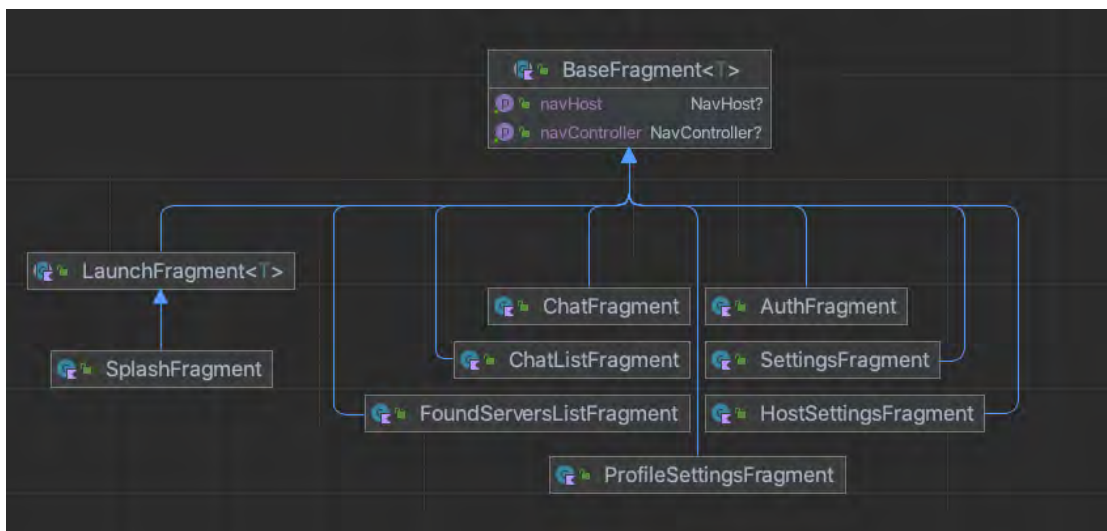


Рисунок 3.14 – Відображення діаграми, що описує наслідування класів-фрагментів від базових

3.2.2 Класи для роботи із інтерактивними повідомленнями

Для роботи із інтерактивними повідомленнями починаючи з версії Android 8.0 потрібно створювати канали. Наявність каналів допомагають відділяти текстові повідомлення від сервісних типів, медійних тощо та гнучко налаштовувати поведінку відображення повідомлень, гучність.

Створено інтерфейс що зберігає усю потрібну інформацію про тип повідомлення, він визначає опис каналу, ідентифікатор іконки, важливість повідомлення та інші параметри.

Створено інтерфейс що зберігає перелік доступних типів – `DeclaredNotificationTypes<T>`, та його реалізацію – `JNotifications`.

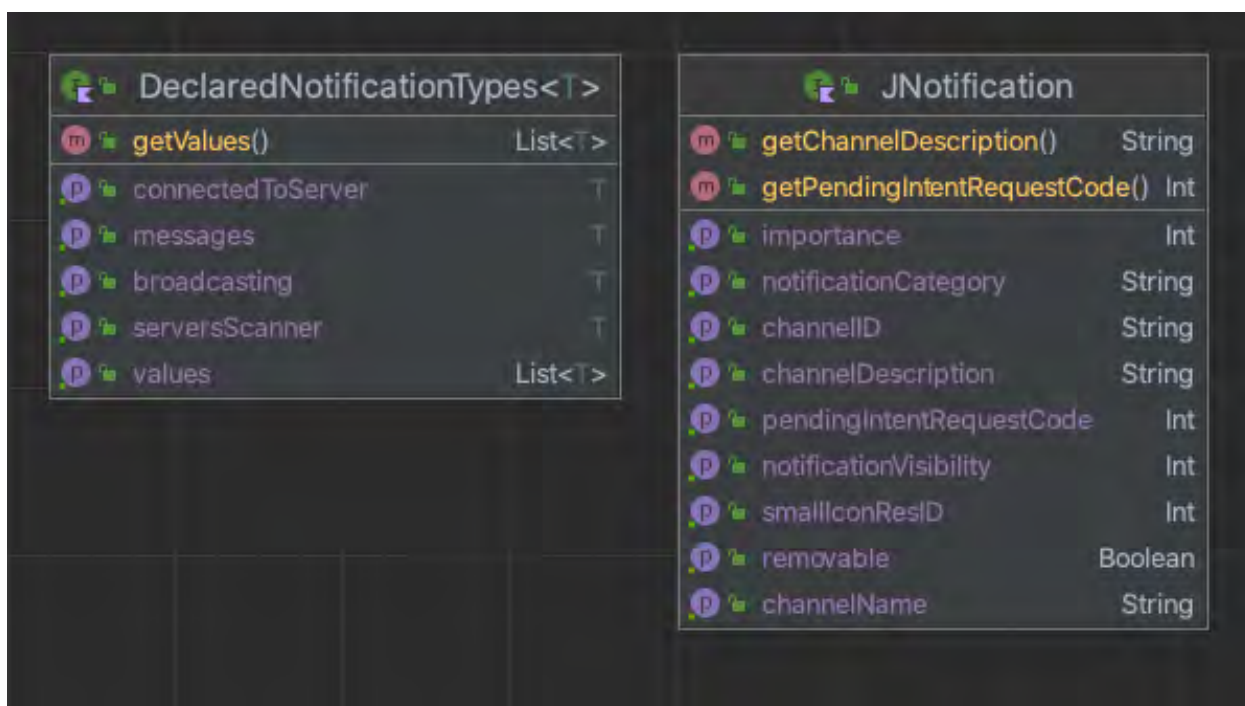


Рисунок 3.15 – Діаграма класів `DeclaredNotificationTypes` та `JNotification`

Для реєстрування каналу у системі Android було написано відповідний алгоритм.

```
// обгортка для функції checkNotificationChannels, що передає
// менеджерПовідомлень та список каналів із розробленого класу
JNotifications
internal fun checkNotificationChannels(context: Context){
    return checkNotificationChannels(
        notificationManager = context.getSystemService(),
        channels = JNotifications.getAsChannels()
    )
}

internal fun checkNotificationChannels(
```

```

notificationManager: NotificationManager,
channels: List<NotificationChannel>
) {
    val installedChannels = notificationManager.notificationChannels
    for (supportedChannel in channels){
        if (!installedChannels.contains { it.id == supportedChannel.id }){
            tryCatch("Channel ${supportedChannel.id}") {

notificationManager.createNotificationChannel(supportedChannel)

            }
        }
    }
}

```

Функції розширення, що перетворюють дані JNotification у android.app.NotificationChannel.

```

fun JNotification.getAsChannel(): NotificationChannel {
    return NotificationChannel(this.channelID, this.channelName,
this.importance).also {
        it.description = this.channelDescription
        it.lockscreenVisibility = this.notificationVisibility
    }
}

fun JNotifications.getAsChannels(): List<NotificationChannel> {
    return this.values.map { it.getAsChannel() }
}

```

3.2.3 Класи, що реалізують збереження та зчитування даних

Усі класи для роботи із збереженням та зчитуванням даних були розроблені за однаковим принципом (рис. 3.16):

- розробка структури даних, які будуть зберігатись на пристрої користувача. (ім'я шаблону – TState);

- розробка інтерфейсу із визначеними методами отримання та встановлення значень;
- розробка класу що наслідується від спроектованого інтерфейсу та класу `Storage<TState>`, що буде обробляти дані визначеної структури.

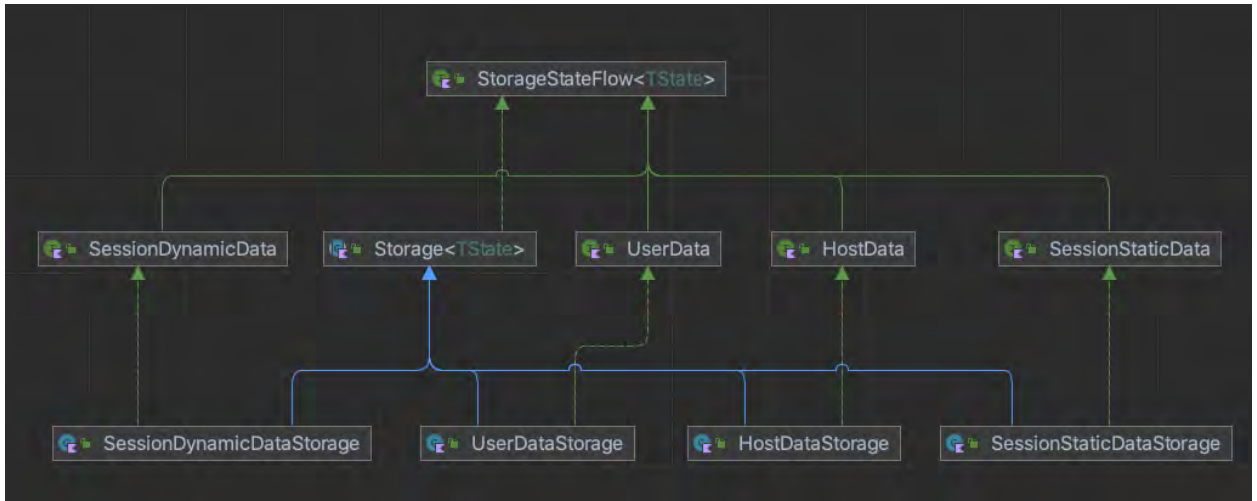


Рисунок 3.16 – Найбільш зрозумілий вигляд діаграми наслідувань усіх класів для роботи із збереженням даних

Наслідування відбувається таким чином:

- усі інтерфейси «*Data» наслідуються від `StorageStateFlow<..>`;
- клас «Storage» наслідується від `StorageStateFlow<..>`;
- усі фінальні класи «*DataStorage» наслідуються від класу «Storage» та інтерфейсу «*Data».

3.2.3.1 Базовий клас `Storage<TState>`

Для зчитування та збереження даних було розроблено базовий клас `Storage<TState>` де `TState` це тип класу, який описує структуру даних що зберігається. `Storage<TState>` наслідує інтерфейс `StorageStateFlow<TState>` що має лише одне поле `state` із типом даних `StateFlow<TState>`. В результаті отримаємо такий вигляд зв'язків наслідувань що зображено на рис. 3.17.

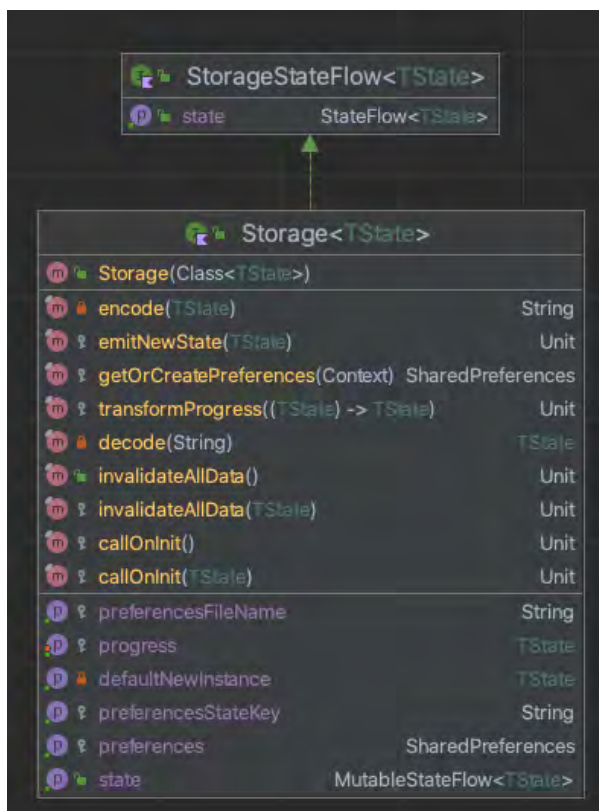


Рисунок 3.17 – Діаграма класу Storage<..> та зв'язок наслідування

До основного функціоналу Storage<TState> відносяться методи getProgress(), setProgress(TState) та transformProgress(TState): TState. Саме ці методи відповідають за зчитування та оновлення даних. Під капотом вони мають таку реалізацію.

```

abstract class Storage<TState>(...): StorageStateFlow<TState> {
    (...)

    protected fun getProgress(): TState {
        return preferences.getString(preferencesStateKey, null)?.let {
            decode(it)
        }?: getDefaultNewInstance()
    }

    protected open fun setProgress(newState: TState){
        preferences.edit().putString(preferencesStateKey, newState?.let {
            encode(it)
        }).apply()
        emitNewState(newState)
    }
}
  
```

```

protected fun transformProgress(transformer: (TState) -> TState){
    val progress = getProgress()
    val copy = transformer(progress)
    setProgress(copy)
}
(...)
}

```

3.2.3.2 Клас HostDataStorage

Для роботи із даними серверу було розроблено інтерфейс HostData, який має увесь потрібний функціонал для роботи із структурою даних HostDataState: отримання та оновлення ідентифікатору та назви серверу, перевірка наявності та додання користувача у списку зареєстрованих.

HostDataState має ідентифікатор та назву серверу, а також масив що зберігає усіх користувачів що були хоча б один раз з'єднані із сервером.

Перелік методів інтерфейсу HostData та моделі класу HostDataState зображено на рис. 3.18.

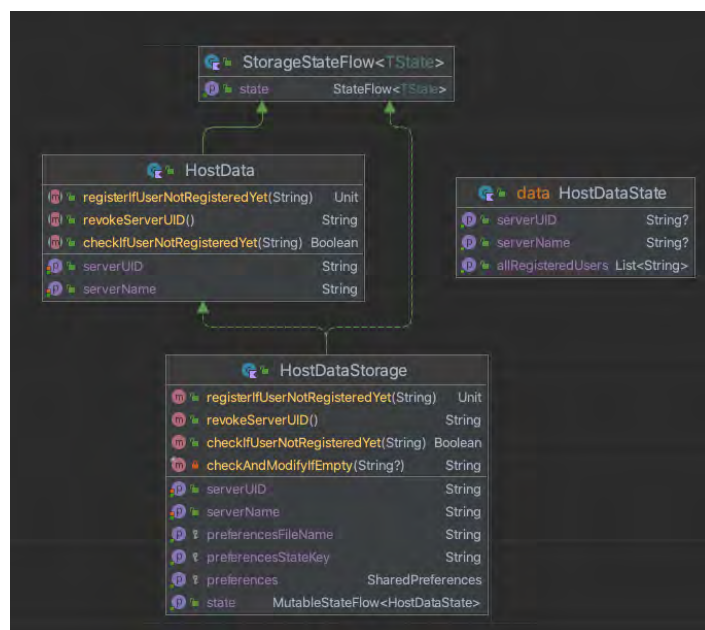


Рисунок 3.18 – Діаграма класу HostDataStorage та класів що наслідуються.

Поруч відображена модель класу HostDataState

3.2.3.3 Клас SessionDynamicDataStorage

Для збереження часто оновлюємих даних було вирішено створити клас SessionDynamicDataStorage, інтерфейс із описаним функціоналом та модель структури даних, що буде зберігатись.

Для ініціалізування об'єкту класу SessionDynamicDataStorage використовується ідентифікатор серверу що надається при підключенні до самого серверу. При ініціалізації створюється файл із назвою «com.edu.alterjuicechat.sessiondynamicdatastorage_{ідентифікатор_серверу}». Це дає змогу зберігати файли із даними що відносяться до конкретного серверу та дає можливість продовжити спілкування у разі повторного підключення до серверу.

Структура даних що зберігається описана у класі SessionDynamicDataState та наведено на рис. 3.19.

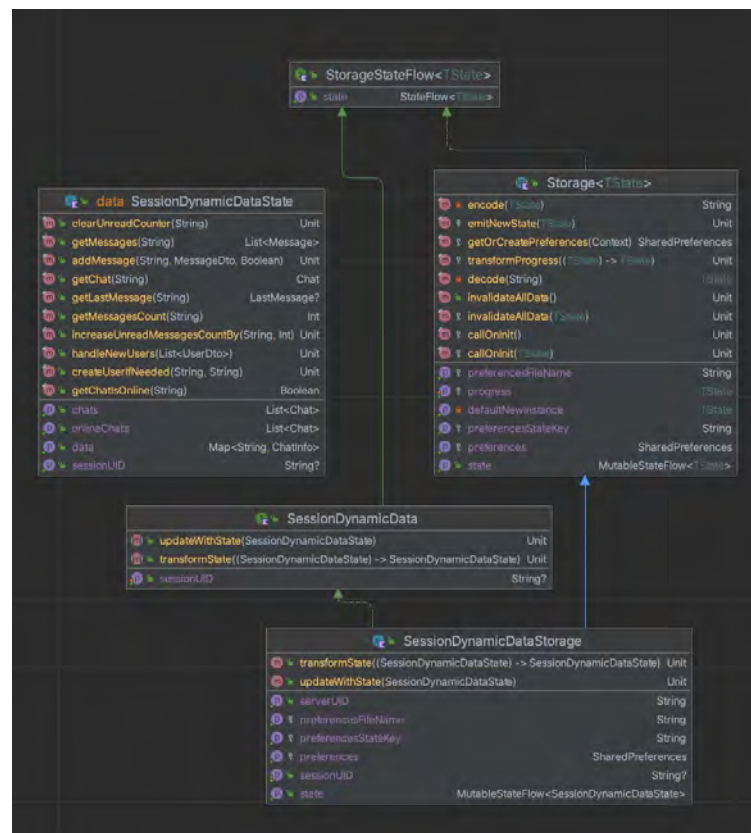


Рисунок 3.19 – Діаграма наслідувань класу SessionDynamicDataStorage та модель даних SessionDynamicDataState

3.2.3.4 Клас SessionStaticDataStorage

Для роботи із статичними даними сесії було створено класи SessionStaticDataStorage, інтерфейс SessionStaticData та модель даних SessionStaticDataState.

Інтерфейс SessionStaticData має методи оновлення та зчитування ідентифікатору поточної сесії. Значення встановлюється як тільки користувач підключається до серверу, та видаляється як пропадає зв'язок (рис. 3.20).

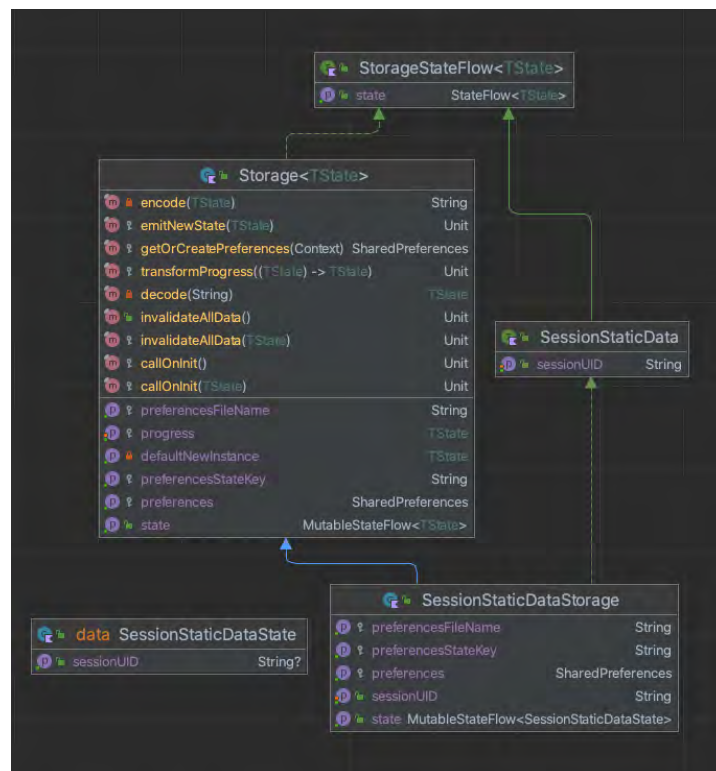


Рисунок 3.20 – Діаграма класу SessionStaticDataStorage

3.2.3.5 Клас UserDataStorage

Для зберігання даних профілю користувача було створено класи UserData, UserDataStorage, UserDataState.

Інтерфейс UserData має лише два методи: встановлення та зчитування імені користувача.

Модель UserDataState має єдине поле – displayName типу String що може бути null (рис. 3.21).

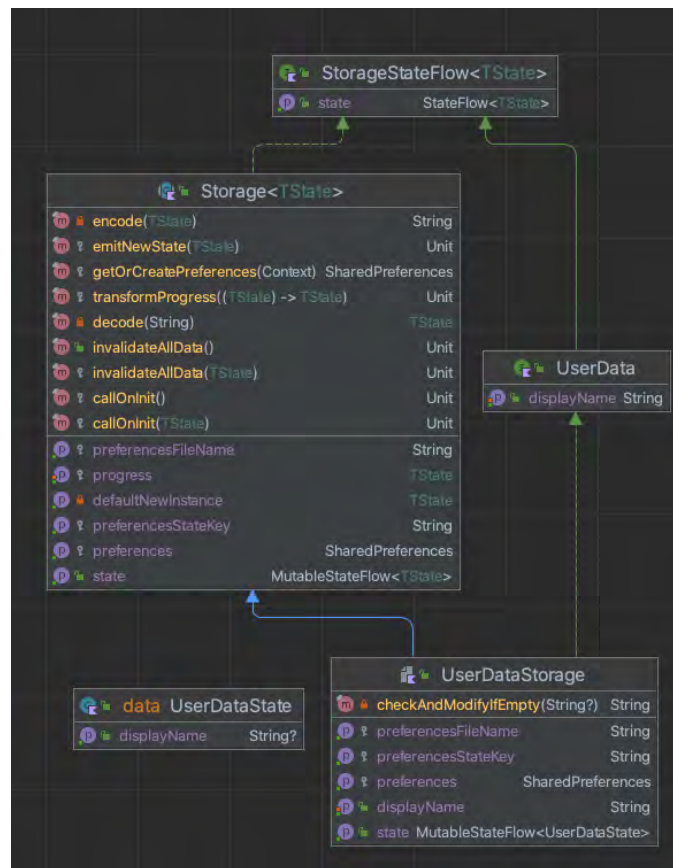


Рисунок 3.21 – Діаграма класів, що використовуються для роботи із даними профілю користувача

3.2.4 Класи, що виконують роль обробки даних з серверу

Для обробки даних було розроблено клас DataRepo, що є об'єктом одинаком у межах однієї сесії користувача. Функціонал класу було вилучено у окремий інтерфейс IDataRepo (рис. 3.22). В основному DataRepo оперує даними із SessionDynamicDataStorage та даними із глобальної видимості – GlobalFlows. Оновлення даних із GlobalFlows сприяє оновленню візуального інтерфейсу користувача у разі підписки на ці дані.

Клас GlobalFlows має такий вигляд.

```

object GlobalFlows {
    val allChats = MutableStateFlow<List<Chat>>(emptyList())
    val activeChatUID = MutableStateFlow<UserUID?>(null)
    val activeChat = MutableStateFlow<Chat?>(null)
    val activeChatMessages = MutableStateFlow<List<Message>>(emptyList())
}

```

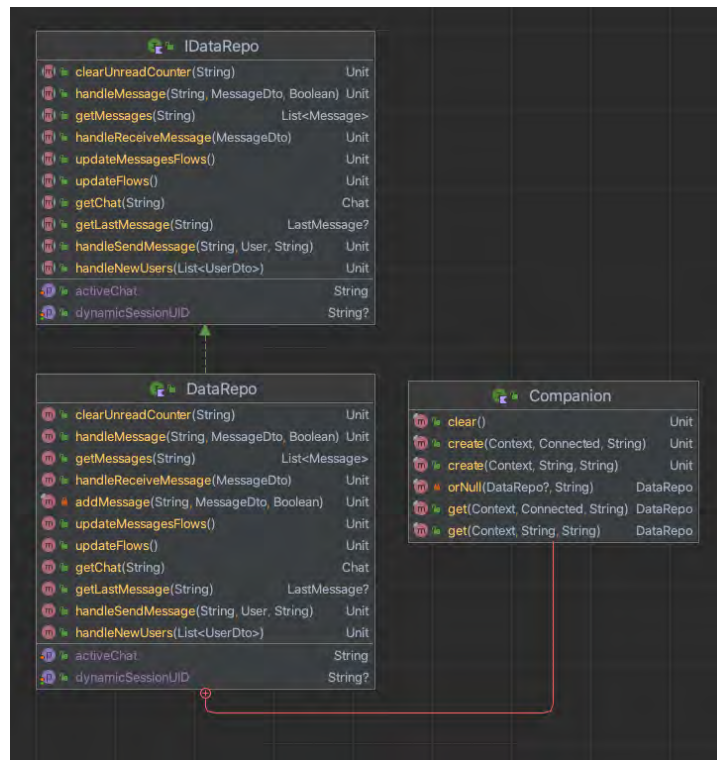


Рисунок 3.22 – Діаграма класу DataRepo

У свою чергу об'єкт dataRepo зберігається у іншому класі глобального доступу – RuntimeConnectivity.

```

object RuntimeConnectivity: RuntimeStatusHolder<ConnectivityStatus>(),
KoinComponent {
    val dataRepoState = MutableStateFlow<IDataRepo>(emptyIRepo)
    (...)
}

```

Також було створено об'єкт emptyIRepo, що є пустою імплементацією IDataRepo. Зроблено це для того щоб міняти значення dataRepoState між пустим та тим, що відноситься до конкретної сесії під час оновлення значення статусу

з'єднання із сервером у класі `RuntimeConnectivity::statusState`. Приклад підписки та оновлення `dataRepoState` всередині класу `RuntimeConnectivity` в залежності від типу значення нового статусу.

```
(...)
statusState.collect { status ->
    if (status is ConnectivityStatus.Connected) {
        (...)
        withContext(Dispatchers.Main) {
            val newDataRepo = DataRepo.get(get<Context>(), status,
            get<UserData>().getDisplayName())
            dataRepoState.emit(newDataRepo)

        }
    } else if (status is ConnectivityStatus.NotConnected){
        withContext(Dispatchers.Main){
            (...)
            dataRepoState.emit(emptyIRepo)
        }
    }
}
(...)
```

3.2.5 Класи підтримки зв'язку між користувачами

Для встановлення зв'язку між користувачами та обміну даними було використано класи `Socket` та `ServerSocket`. Для коректної роботи (приймання та відправлення даних) було реалізовано наступні описані класи а також моделі даних якими спілкуються сервер та клієнт (рис. 3.23).

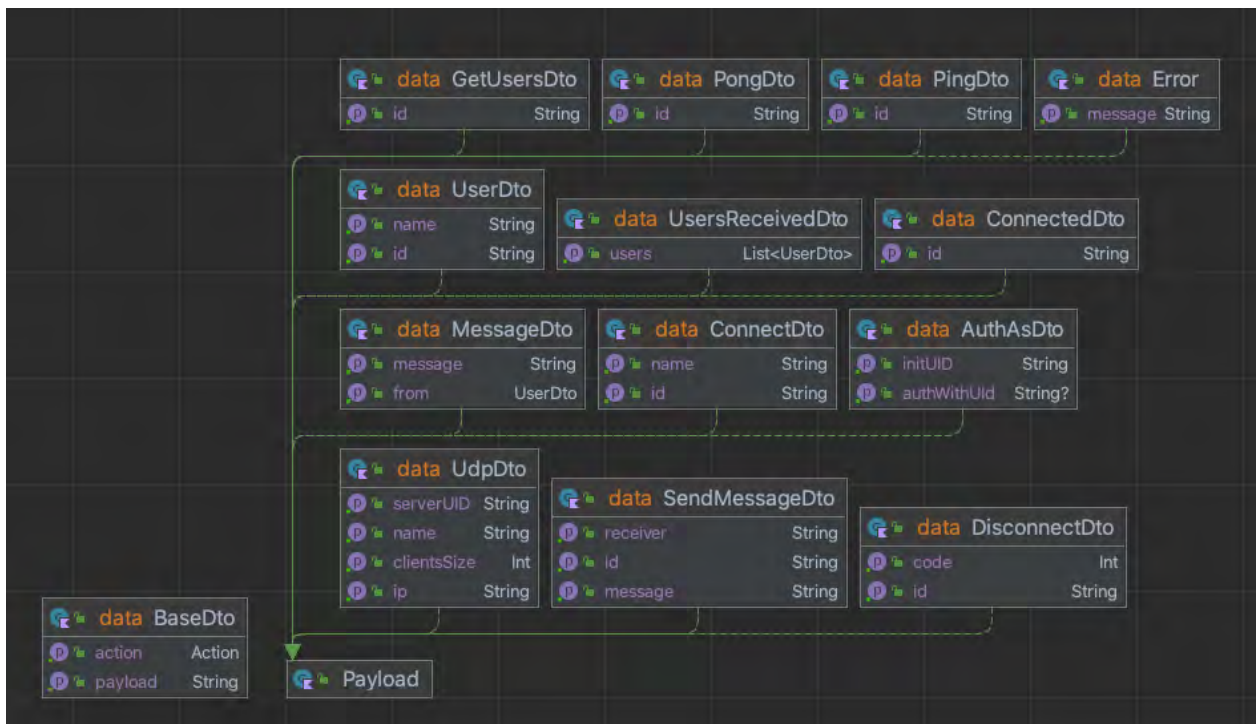


Рисунок 3.23 – Відображення моделі даних, якими спілкуються клієнт та сервер

Опис кожної моделі даних.

PingDto – об’єкт, що періодично надсилається до серверу задля перевірки роботи сервер та інформування, що пристрій має зв’язок.

PongDto – об’єкт, що повертається у вигляді відповіді на PingDto сервером.

ConnectedDto – об’єкт, що повертається після першого з’єднання клієнта із сервером та містить ідентифікатор користувача.

ConnectDto – об’єкт, що надсилається до серверу із вказаним ідентифікатором та іменем користувача.

AuthAs – об’єкт, що відправляє клієнт до серверу, у разі якщо у файлах даних застосунку знайдеться остання збережена сесія.

DisconnectDto – об’єкт, що відправляється клієнтом до серверу для роз’єднання із сервером.

GetUsersDto – команда для отримання переліку усіх користувачів.

MessageDto – об’єкт, що описує повідомлення: текст, та об’єкт UserDto.

SendMessageDto – відправка повідомлення вказаному користувачу за ідентифікатором.

UserDto – об’єкт, що описує користувача: вміщує ім’я та ідентифікатор.

UsersReceivedDto – об’єкт, що відправляється сервером до клієнту та містить перелік користувачів що у поточний час з’єднані із сервером.

UdpDto – об’єкт, що відправляє сервер до клієнту та вміщує поля що описують сервер: назва та адреса, а також кількість користувачів «онлайн».

3.2.4.1 JClient – клас для підключення до серверу

JClient представляє собою клас, що містить поля типу Socket та функцію зворотного виклику, що обробляє нові дані отримані сокетом. Для встановлення зв’язку викликається метод initialize у класі JClient який приймає IP адресу та порт отриманий під час сканування доступних серверів.

```
private fun recreateSockets(scope: CoroutineScope, ip: String, port: Int){
    socket?.close()
    jSocketHandler?.disconnect()
    socket = Socket().also {
        it.connect(InetSocketAddress(ip, port))
        jSocketHandler = JSocketHandler(it, defaultListener, scope).also {
handler ->
            handler.loop()
        }
    }
}
```

Відправка даних до серверу.

```
fun send(message: String) {
    jSocketHandler?.send(message)
}
```

Визволення ресурсів відбувається наступним чином.

```
fun awaitAndClose() {  
    isRunning = false  
    jSocketHandler?.disconnect()  
    socket?.close()  
}
```

3.2.4.2 ClientWorker – клас, що є обгорткою для JClient

Клас ClientWorker (рис. 3.24) виконує обробку отриманих з серверу даних через функцію зворотного виклику що передається при ініціалізуванні JClient.

```
private val client = JClient(this::onNewPayload)
```

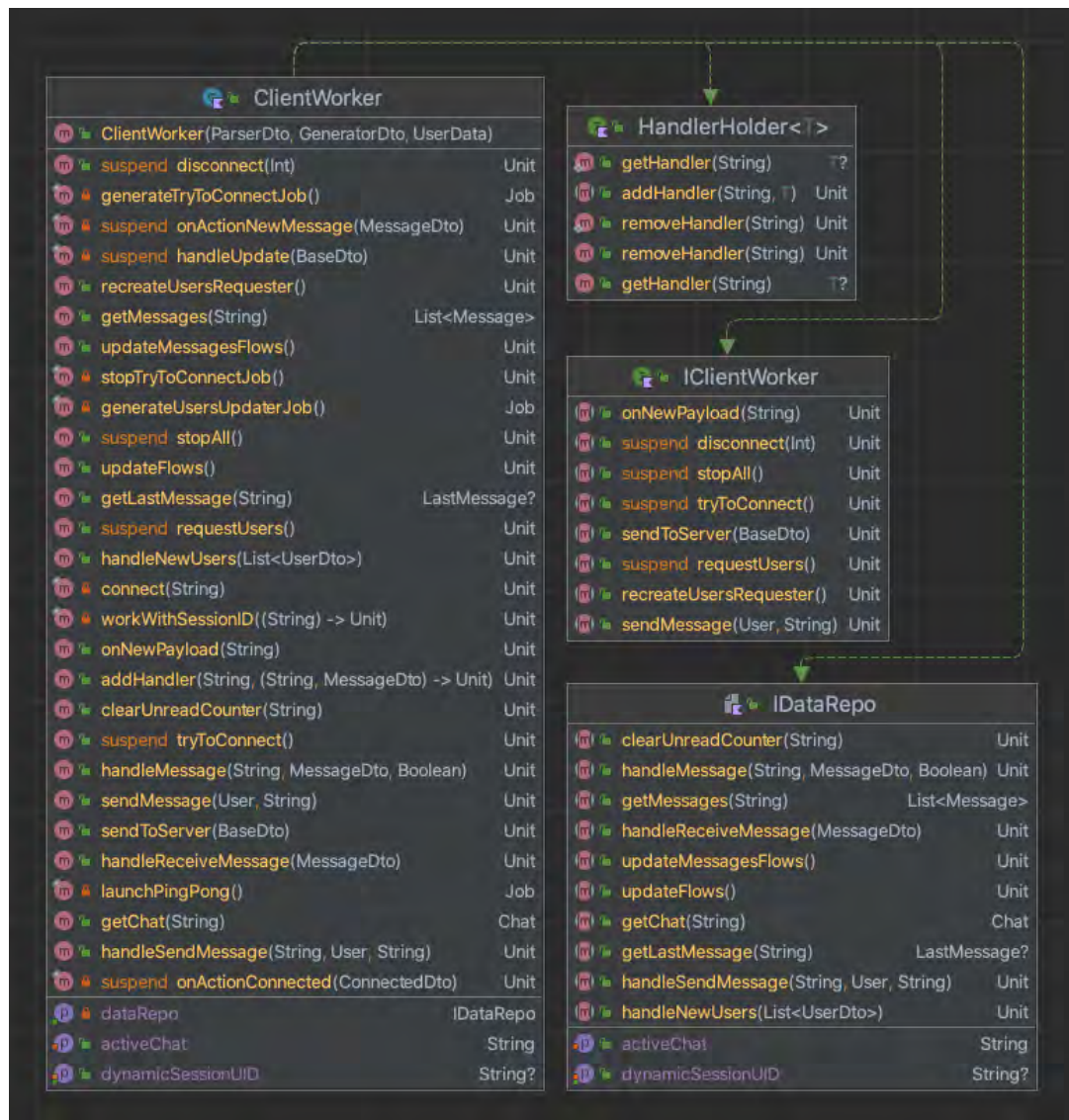


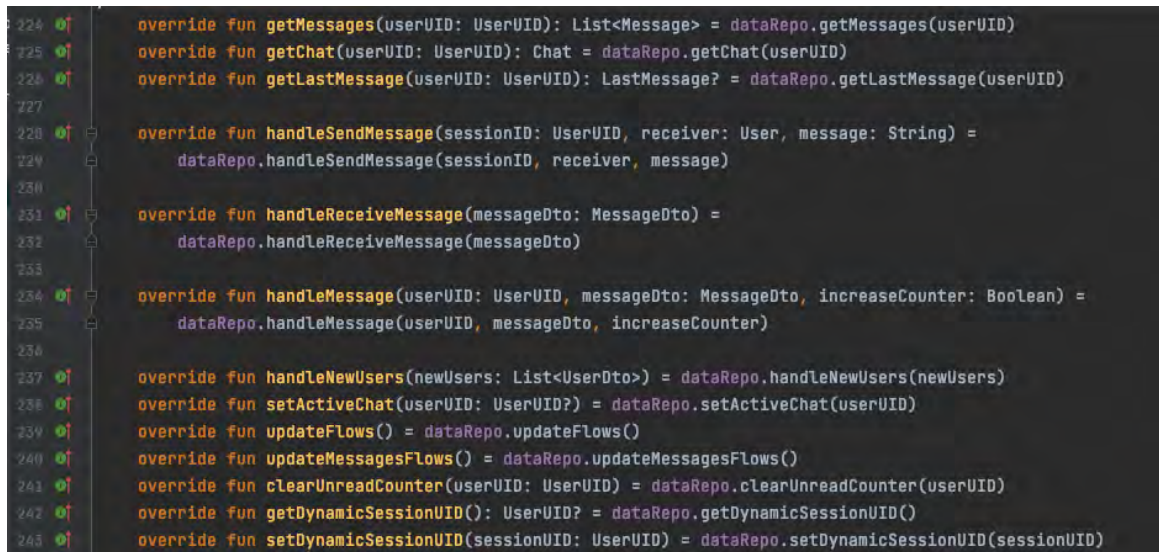
Рисунок 3.24 – Діаграма наслідувань класу ClientWorker

Також клас має свойство `dataRepo` типу `IDataRepo`, що отримується із `RuntimeConnectivity.dataRepoState.value`.

```
private val dataRepo get() = RuntimeConnectivity.dataRepoState.value
```

Так як це свойство, то значення отримується при кожному звертанні до поля `dataRepo`, адже `RuntimeConnectivity::dataRepoState` може змінюватись під час роботи застосунку як на пусту реалізацію (`emptyIRepo`), так і на ту, що зберігає дані для роботи із поточною сесією користувача. Такий підхід розглянуто у п. 3.2.4 про `emptyIRepo`.

У якомусь сенсі ClientWorker є класом-декоратором бо повторює функціонал IDataRepo, як і клас DataRepo (рис. 3.25).



```

224 override fun getMessages(userID: UserID): List<Message> = dataRepo.getMessages(userID)
225 override fun getChat(userID: UserID): Chat = dataRepo.getChat(userID)
226 override fun getLastMessage(userID: UserID): LastMessage? = dataRepo.getLastMessage(userID)
227
228 override fun handleSendMessage(sessionID: UserID, receiver: User, message: String) =
229     dataRepo.handleSendMessage(sessionID, receiver, message)
230
231 override fun handleReceiveMessage(messageDto: MessageDto) =
232     dataRepo.handleReceiveMessage(messageDto)
233
234 override fun handleMessage(userID: UserID, messageDto: MessageDto, increaseCounter: Boolean) =
235     dataRepo.handleMessage(userID, messageDto, increaseCounter)
236
237 override fun handleNewUsers(newUsers: List<UserDto>) = dataRepo.handleNewUsers(newUsers)
238 override fun setActiveChat(userID: UserID?) = dataRepo.setActiveChat(userID)
239 override fun updateFlows() = dataRepo.updateFlows()
240 override fun updateMessagesFlows() = dataRepo.updateMessagesFlows()
241 override fun clearUnreadCounter(userID: UserID) = dataRepo.clearUnreadCounter(userID)
242 override fun getDynamicSessionUID(): UserID? = dataRepo.getDynamicSessionUID()
243 override fun setDynamicSessionUID(sessionUID: UserID) = dataRepo.setDynamicSessionUID(sessionUID)

```

Рисунок 3.25 – Частина коду із класу ClientWorker, що викликає функціонал DataRepo

Також ClientWorker запускає корутину, що із періодичністю пінгує сервер (відправляє PingDto об'єкт).

```

private fun launchPingPong() = scope.launch(Dispatchers.IO) {
    while (true) {
        workWithSessionID { finalSessionID ->
            sendToServer(generator.generatePing(finalSessionID))
        }
        delay(Random.nextLong(2000, 4500))
        // Random time to wait
    }
}

```

3.2.4.3 JServer – клас що є сервером та центром обміну даних між користувачами

На рисунку 3.26 зображена діаграма класу JServer.

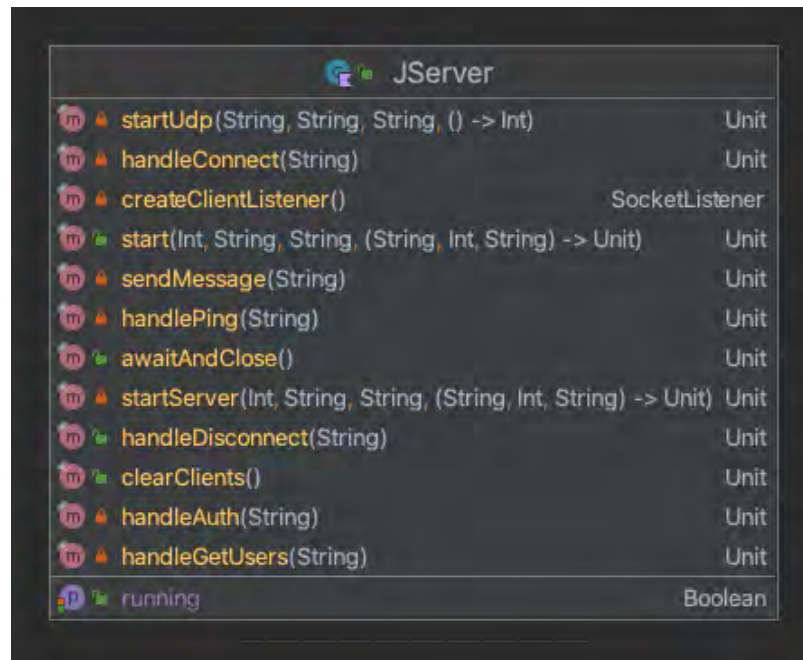


Рисунок 3.26 – Діаграма класу JServer

Для створення серверу було використано клас ServerSocket. Його робота дуже проста. Алгоритм запуску серверу.

```
server = ServerSocket(port).also {
    it.reuseAddress = true
}
val s = DatagramSocket()
s.connect(InetAddress.getByName("8.8.8.8"), 10002)
val address = s.localAddress
val serverIP = address.hostAddress
```

Такі отримані дані як адреса використовуються для віщування через UDP, за допомогою DatagramSocket. Як тільки клієнт буде відсилати запити через UDP, його буде обробляти сервер та повертати інформацію про себе через

об'єкт `UdpDto`. `UdpDto` вміщує поля адреси, імені серверу та ідентифікатор а також кількість клієнтів. Відправка `UdpDto` об'єкту виконується наступним чином.

```
val buffer = ByteArray(256)
var packet = DatagramPacket(buffer, buffer.size)
udpServer?.receive(packet)
val received = packet.data.decodeToString()
println("##### HOST MSG: udp received: ${received.slice(0..10)}")
val response = gson.toJson(UdpDto(serverIP, serverName, serverUID,
    getClientsSize()))
    .toByteArray()
udpServer?.send(
    DatagramPacket(
        response,
        response.size,
        packet.address,
        packet.port
    )
)
```

Після отримання клієнтом таких даних налаштовується сокет та запускається підключення до серверу.

Разом із запуском серверу відбувається прослуховування вхідних даних. Вхідні дані отримуються у вигляді JSON об'єкту що потім декодуються методом `convertPayloadTo<T>`, де `T` це тип класу `*Dto`, що визначені у модулі `com.edu.alterjuicechat.server_model.dto.entities`.

```
val dto = convertPayloadTo<BaseDto>(message)
```

Вигляд функції декодування.

```
private inline fun <reified T> convertPayloadTo(payload: String): T {
    return gson.fromJson(payload, T::class.java)
}
```

Обробка даних відбувається наступним чином.

```
private fun createClientListener(): SocketListener = object : SocketListener {
    override fun onNewMessage(message: String) {
        try {
            val dto = convertPayloadTo<BaseDto>(message)
            when (dto.action) {
                AUTH_AS -> {
                    handleAuth(dto.payload)
                }
                PING -> {
                    handlePing(dto.payload)
                }
                CONNECT -> {
                    handleConnect(dto.payload)
                }
                SEND_MESSAGE -> {
                    sendMessage(dto.payload)
                }
                GET_USERS -> {
                    handleGetUsers(dto.payload)
                }
                DISCONNECT -> {
                    handleDisconnect(dto.payload)
                }
                else -> println("unknown action: $message")
            }
        } catch (e: Exception) {
            println("error handling message: ${e.message}")
        }
    }
}
```

Приклад відправлення повідомлення сервером до клієнту.

```
private fun sendMessage(payload: String) {
    val sendData = convertPayloadTo<SendMessageDto>(payload)
    clients.firstOrNull { it.user.id == sendData.receiver }?.also { receiver ->
        clients.firstOrNull { it.user.id == sendData.id }?.also { sender ->
            val message = MessageDto(sender.user, sendData.message)
        }
    }
}
```

```

        receiver.jSocketHandler.send(gson.toJson(BaseDto(NEW_MESSAGE,
gson.toJson(message))))
    }
}
}

```

Алгоритм приєднання користувача до серверу.

- отримання даних з UDP про серверу;
- підключення до серверу та отримання ідентифікатору користувача;
- якщо користувач вже авторизувався на цьому сервері, то відправляє команду заміни ідентифікатору на старий разом з новим;
- сервер порівнює чи є у системі збережений старий ідентифікатор. Якщо так то замінює його на переданий користувачем та відправляє відповідь клієнту із фінальним ідентифікатором;
- користувач вважається авторизованим після отримання фінального ідентифікатору.

3.2.5 Розроблені сервіси, що працюють у фоновому режимі

Для роботи із сервісами було розроблено базовий клас який допомагає ефективніше працювати з тими компонентами, що будуть наслідувати його.

Роль цього класу – створення окремого CoroutineScope та додаткового функціоналу, що допоможе без помилок зупиняти працюючі сервіси. Основний метод – це manuallyStop, що керує та визволяє ресурси, що більше не потрібні. Модель базового класу та класів що наслідують його зображено на рисунку 3.27.

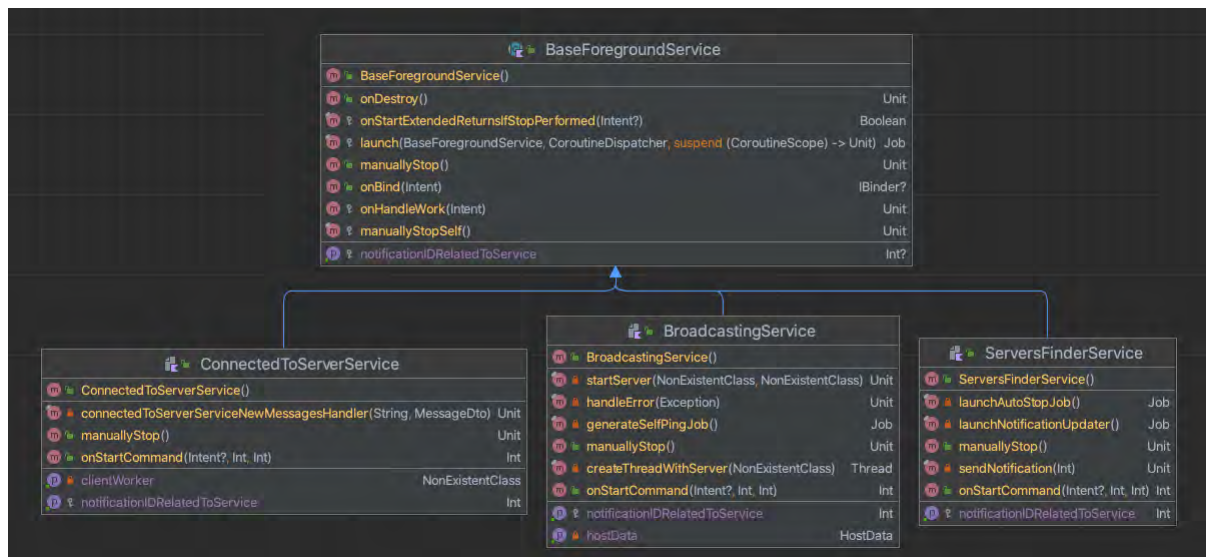


Рисунок 3.27 – Діаграма класу BaseForegroundService та класів що наслідують його

3.2.5.1 Сервіс пошуку мережових кімнат(ServersFinderService)

Для пошуку мережових кімнат було створено сервіс ServersFinderService задача якого стартувати пошук та показувати інтерактивні повідомлення, що оновлюються кожного разу як знаходяться нові сервери поблизу.

Із стартом сервісу водночас стартує пошук серверів командою finder.startListening().

```

override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    val performedStop = onStartExtendedReturnsIfStopPerformed(intent)
    if (performedStop){
        return Service.START_NOT_STICKY
    }
    if (!finder.serverScannerIsWorking.value){
        launch {
            finder.startListening()
        }
    }
    autoStopJob.renew()
    notificationUpdater.renew()
    val notification =
ServersScannerNotificationGenerator.createNotification(this, 0)
  
```

```

        startForeground(getNotificationIDRelatedToService(), notification.build())
        return Service.START_NOT_STICKY
    }

```

Також було передбачено автоматичне зупинення сервісу. Це реалізовано за допомогою корутин, що запускають роботу у якій першою командою є зупинка співпрограми на одну хвилину, а після цього виконується виклик методу `manuallyStopSelf()` базового класу.

```

private fun launchAutoStopJob() = launch {
    delay(60*1_000)
    manuallyStopSelf()
}
override fun manuallyStop() {
    launch { finder.stopListening() }
    notificationUpdater.cancelIt()
    autoStopJob.cancelIt()
    super.manuallyStop()
}

```

Логіка пошуку серверів описана у класі `ServersFinderImpl`, що наслідуює `ServersFinder` інтерфейс (рис. 3.28). Для його роботи використовуються клас `DatagramSocket()`, що відправляє масив байтів на адресу «255.255.255.255». У цей час, якщо інший користувач увімкнув режим хосту, то його пристрій отримає цей пакет даних, та відправить йому відповідну інформацію про себе: IP адреса, порт, назва серверу та кількість клієнтів. Далі ці дані опрацьовуються та зберігається у об'єкт типу `MutableStateFlow<..>` визначений у інтерфейсі `ServersFinderState`. Безпосередньо у вікні та у сервісі `ServersFinderService` спрацьовує функція зворотного виклику, та відповідно реагує на ці оновлення. Наприклад – відбувається встановлення списку доступних серверів, та оновлюється кількість знайдених серверів в інтерактивному повідомленні. Код роботи пошуку серверів.

```

private fun listenToServers(
    udpPort: Int = ServerConsts.UDP_PORT,
): Job {
    return scope.launch(Dispatchers.IO) {
        var attempts = 0
        val messageToSend =
ServerConsts.SERVER_UDP_KEY_TO_GET_TCP_IP.encodeToByteArray()
        val datagramSocket = DatagramSocket().apply { soTimeout =
ServerConsts.UDP_TIMEOUT }
        val receiveBuffer = ByteArray(ServerConsts.UDP_PACKET_SIZE)
        val receivePacket = DatagramPacket(receiveBuffer, receiveBuffer.size)
        var udpAddress: String
        while (serverScannerIsWorking.value) {
            delay(1000)
            attempts += 1
            udpAddress = ServerConsts.UDP_ADDRESS_BOTH.let { it[attempts %
it.size] }
            try {
                val sendPacket = DatagramPacket(
                    messageToSend,
                    messageToSend.size,
                    InetAddress.getByName(udpAddress),
                    udpPort
                )
                datagramSocket.send(sendPacket)
                datagramSocket.receive(receivePacket)
                val udpPayloadStr = String(receivePacket.data, 0,
receivePacket.length)
                val serverInfo = ServerInfo(parser.parseUdp(udpPayloadStr))
                servers[serverInfo.ip] = serverInfo
                println("ServersFinder: UDP found! Server: ${serverInfo};")
                availableServers.emit(servers.values.toList())
            } catch (timeout: SocketTimeoutException) {
                println("ServersFinder: <SocketTimeoutException>:
${timeout.message}")
                delay(1_000)
            } catch (e: IOException) {
                e.printStackTrace()
                println("ServersFinder: <IOException>: ${e.message}")
                delay(1_000)
            }
        }
    }
}

```

}

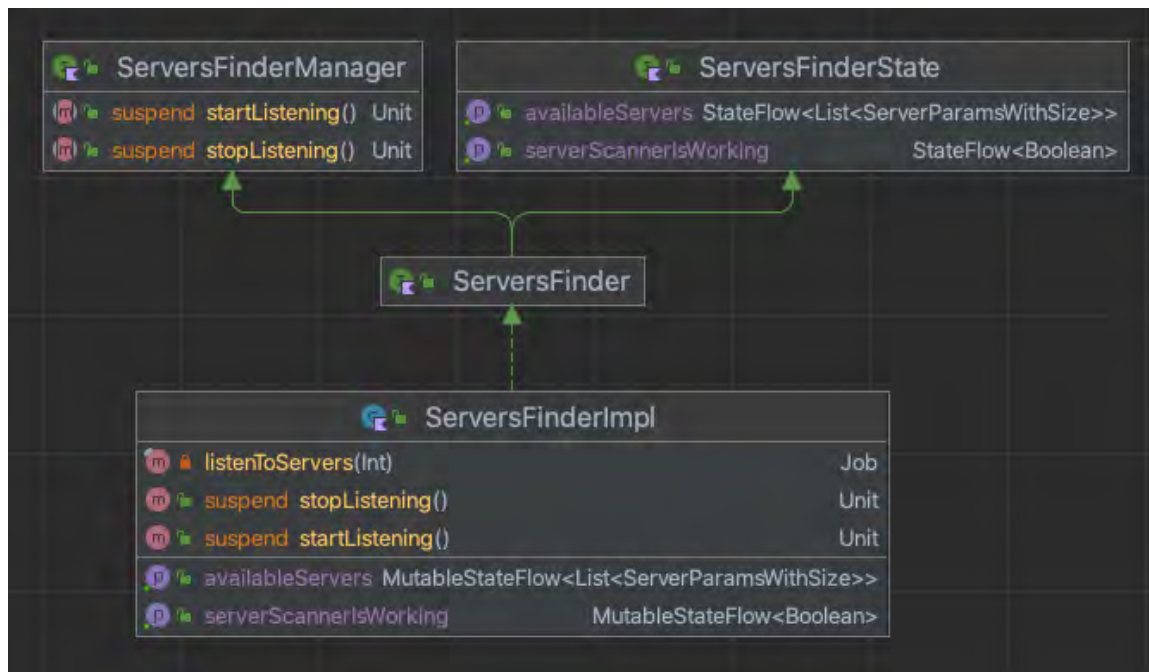


Рисунок 3.28 – Діаграма класів та їх функціонал, що наслідує ServersFinderImpl

3.2.5.2 Сервіс що запускає роботу хосту (BroadcastingService)

Робота цього сервісу передбачає запуск самого серверу та віщування UDP, що при з'єднанні із пристроєм відправляє дані про сервер – об'єкт UdpDto.

При ініціалізуванні сервісу створюється інтерактивне повідомлення, що інформує що сервіс працює та може бути знайдений іншими користувачами підключеними до однієї мережі.

Для роботи створюється новий потік та ініціалізується об'єкт JServer. Також при ініціалізуванні передається функція зворотного виклику «onServerStarted», що викликається коли сервер починає свою роботу.

```

private val onServerStarted = OnServerStart { ip: String, port: Int, name:
String ->
    // Створення повідомлення про старт сервісу
    startForeground(
        getNotificationIDRelatedToService(),

```

```

BroadcastingNotificationGenerator.createNotification(this@BroadcastingService,
ip, name).build()
    )
    launch {
        delay(1500)
        // Встановлення нового значення стану серверу
        RuntimeHost.emitNewStatus(HostStatus.Broadcasting(ip, port))
    }
}

```

Функція, що викликається при ініціалізації сервісу.

```

override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
    val stopPerformed = onStartExtendedReturnsIfStopPerformed(intent)
    if (stopPerformed) {
        return Service.START_NOT_STICKY
    }
    startForeground(getNotificationIDRelatedToService(),
BroadcastingNotificationGenerator.createNotificationWithoutText(this).build())
    selfPingJob.renew()
    // Встановлення значення стану роботи серверу на «HostStatus.Preparing»
    RuntimeHost.emitNewStatus(HostStatus.Preparing)

    serverThread = Thread {
        server = JServer(
            checkAndAddIfUserUIDNotRegisteredYet =
hostData::checkIfUserNotRegisteredYet,
            registerUserUID = hostData::registerIfUserNotRegisteredYet
        ).also {
            try {
                it.start(
                    port = ServerConsts.SERVER_PORT,
                    serverName = hostData.getServerName(),
                    serverUID = hostData.getServerUID(),
                    onStart = onServerStarted
                )
            } catch (e: Exception) {
                handleError(e)
            }
        }
    }
}

```

```

    }.also {
        it.start()
    }
    return Service.START_NOT_STICKY
}

```

3.2.5.3 Сервіс підключення до серверу (ConnectedToServerService)

Сервіс підключення до серверу служить для підтримки зв'язку у моменти згортання застосунку до фоновому режиму. Також сервіс створює інтерактивне повідомлення що відображає статус з'єднання.

Робота сервісу проста. Створюється об'єкт класу `ClientWorker` та викликається метод `tryToConnect`.

```

launch {
    clientWorker.tryToConnect()
}

```

У свою чергу метод `ClientWorker::tryToConnect` перезапускає роботу циклу, що перевіряє статус підключення. Якщо статус підключення є «`ConnectivityStatus.ChosenServerToConnect`», то визивається метод `JClient::initialize`, що вже створює та налагоджує підключення до вказаної IP адреси та порту за допомогою сокетів. Опис класу `JClient` розглянуто у п. 3.2.4.1.

Паралельно із цим запускається нова корутина, що у циклі `while` перевіряє статус з'єднання із відліком часу в одну секунду. Якщо статус змінився, то оновлюється інтерактивне повідомлення із актуальним статусом, а користувача повертає на головний екран.

Також під час ініціалізації додається функція зворотного виклику що викликається при отриманні нового повідомлення. Це використовується для додаткової обробки та створення текстових повідомлень, що будуть показуватись у менеджері сповіщень.

```
private fun connectedToServerServiceNewMessagesHandler(activeChatUID: UserID?,
newMsg: MessageDto){
    if (activeChatUID != newMsg.from.id) {
        // Створення та генерація повідомлення на основі даних що отримано
        MessageNotificationGenerator.createAndSendNotification(
            this,
            newMsg.toNotificationMessageWithID()
        )
    }
}
```

3.2.6 Класи та функції утиліти, що були розроблені

3.2.6.1 RenewableJob

RenewableJob це клас, що у якості аргументу отримує функцію, що повертає об'єкт типу Job. Такий клас було створено задля вирішення проблеми дублювання коду відміни створеної корутинами роботи та повторного їх запуску (рис. 3.29).

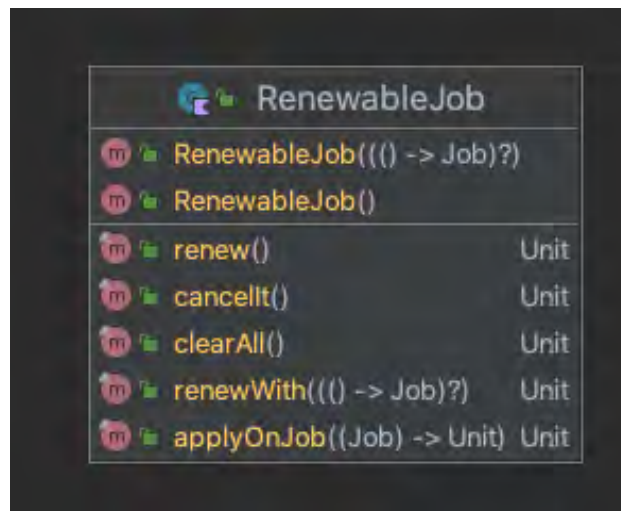


Рисунок 3.29 – Діаграма класу RenewableJob

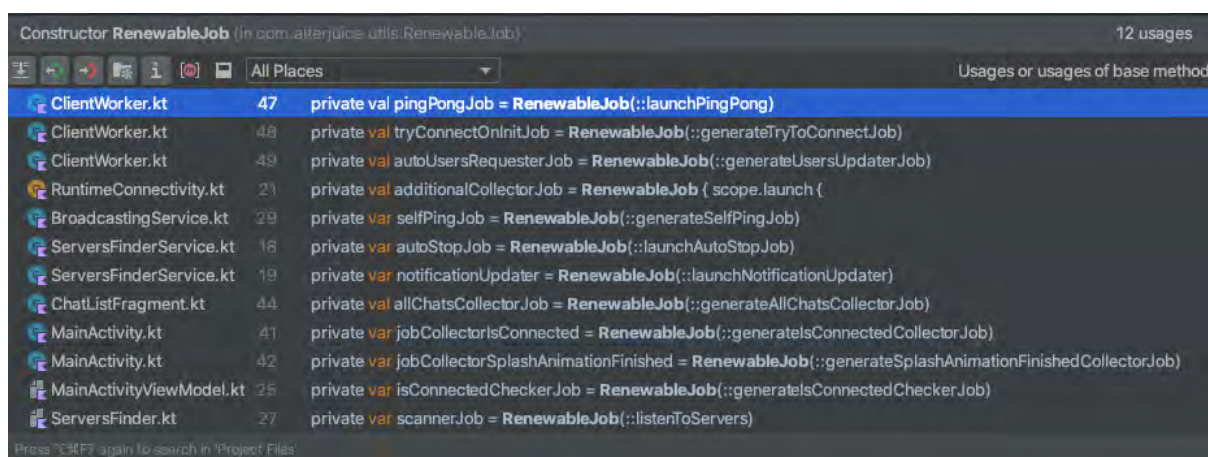


Рисунок 3.30 – Перелік місць використання RenewableJob

Основні його методи це `cancelIt()` та `renew()`. `CancelIt()` зупиняє роботу, що виконувалась, а `renew()` повторно запускає роботу через виклик функції, що передавалась при ініціалізації об'єкту. Приклад роботи такого об'єкту можна побачити у сервісах, вьюмоделях тощо (рис. 3.30). Код автоматичного зупинення сервісу підключення до серверу:

Кожні 2,5 секунди викликається функція, що відмінює роботу `selfTurnerOff` та починає її заново. Якщо робота `selfTurnerOff` не буде відмінена, то за відліком 10 секунд змінює значення статусу хоста на «Disabled».

```
// ініціалізування об'єкту RenewableJob із посиланням на generateSelfPingJob
private var selfPingJob = RenewableJob(::generateSelfPingJob)
private fun generateSelfPingJob() = launch(Dispatchers.IO) {
    while (true){
        RuntimeHost.notifySelfPinged()
        delay(2_500)
    }
}
```

Функція `RuntimeHost.notifySelfPinged()`.

```
object RuntimeHost: RuntimeStatusHolder<HostStatus>() {
    (...)
    override fun notifySelfPinged() {
```

```

        recreateSelfTurnerOff()
    }

    override fun recreateSelfTurnerOff(){
        selfTurnerOff?.cancel()
        selfTurnerOff = scope.launch {
            delay(10_000) // відлік 10 секунд
            if (statusState.value is HostStatus.Broadcasting) {
                statusState.emit(HostStatus.Disabled)
            }
        }
    }
    (...)
}

```

3.2.6.2 RuntimeStatusHolder

Для збереження статусів з'єднань було створено клас, що надає увесь потрібний функціонал для роботи оновлення цих даних (рис. 3.31).

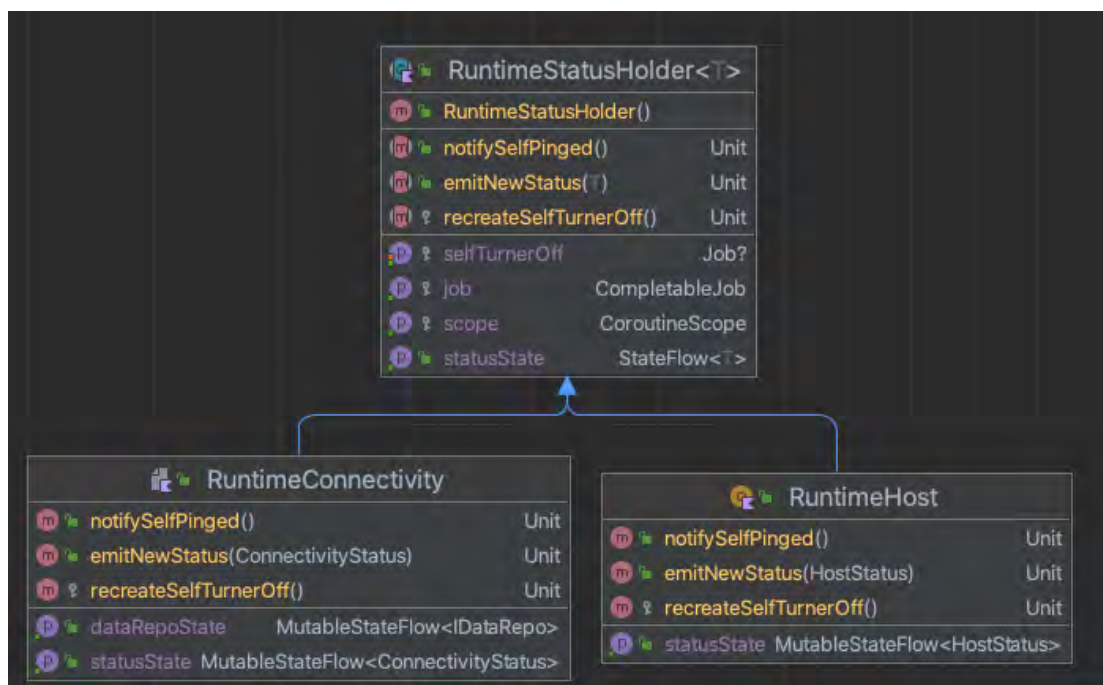


Рисунок 3.31 – Модель абстрактного класу `RuntimeStatusHolder<T>` та класів що наслідують його

Такі класи як `RuntimeActivity` та `RuntimeHost` є класами-сінглтонами, та надають єдине значення статусів під час роботи застосунку. Інші класи можуть звертатись до поля `statusState` та підписуватись на зміни значення. Також наявний функціонал автоматичної зміни значення, у разі якщо сервіси не інформують про те що вони активні. Таку реалізацію можна продивитись у класах `BroadcastingService` та `ConnectedToServerService`, що використовують метод `notifySelfPinged()`.

3.2.6.3 Функції-розширення

Функція генерування макету при використанні технології `ViewBinding`.

```
inline fun <T : ViewBinding> AppCompatActivity.viewBinding(crossinline
bindingInflater: (LayoutInflater) -> T) =
    lazy(LazyThreadSafetyMode.NONE) {
        bindingInflater.invoke(layoutInflater)
    }
```

Створення об'єкту `ViewBinding` у акті віті.

```
private val binding by viewBinding(ActivityMainBinding::inflate)
```

Функція поділу числа на 2.

```
fun Int.half(): Float = this / 2f
```

Функція анімованого збільшення елемента інтерфейсу.

```
fun View.scaleUpDown(increaseBy: Float = 1.1f, duration: Long = 200){

    this.animate().scaleY(increaseBy).scaleX(increaseBy).setDuration(duration).withE
ndAction{this.animate().scaleX(1f).scaleY(1f).start()}
}
```

Функція перевірки одного об'єкту до прилежності до одного з типів зазначеного у списку переданого як аргумент.

```
infix fun <T: R, R> T.isOneOf(types: List<Class<out R>>): Boolean {
    val contextClass = this!!::class.java
    types.forEach {
        if (it == contextClass){
            return true
        }
    }
    return false
}
```

3.3 Огляд візуального інтерфейсу застосунку

3.3.1 Основні рішення щодо дизайну застосунку

Основним підходом до розробки інтерфейсу було використано «Material Design». Це вважається сучасним підходом та стандартом в розробці Android застосунків. Суть Material Design в тому, щоб використовувати стиль елементів подібний до карток. Додається тінь, що створює ілюзію плаваючих елементів на екрані. На поточний момент вже доступний новий підхід – «Material 3». Його ідея полягає в тому, щоб позбутися заздалегідь встановлених кольорів додатку, та використовувати системні кольори або такі, що залежать від улюбленої кольорової гами користувача.

Також було створено кастомний елемент інтерфейсу WiFiWavesView, що відображає анімовані хвилі. Для більш детального ознайомлення див. пункт 3.3.2.

Ще однією деталлю є використання анімованого фону додатку. Це градієнт, що переливається із голубого до білого кольору, та навпаки. Це майже непомітно, але дає відчуття приємної роботи із інтерфейсом.

3.3.2 Кастомний елемент інтерфейсу – WiFiWavesView

Для більш приємної та живої роботи застосунку було вирішено створити кастомний елемент інтерфейсу, а саме – анімовані хвилі, що за дизайном розміщуються за кнопкою круглої форми. Ці хвилі символізують роботу модуля Wi-Fi. Було розроблено функціонал, який допомагає керувати хвилями, показувати або сховувати їх.

```
interface WiFiWavesController {
    fun showWaves()
    fun hideWaves()
    fun isVisible(): Boolean
    fun showHideWaves(isVisible: Boolean)
}
```

Також розроблений елемент інтерфейсу має багато параметрів для гнучкого налаштування (рис. 3.32).

```
<com.edu.alterjuicechat.ui.custom.wifi_waves.WiFiWavesView
    android:id="@+id/scanningServersWaves"
    style="@style/WiFiWavesView"
    android:layout_width="match_parent"
    android:layout_height="350dp"
    android:layout_gravity="center"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"

    app:drawBy="wavesAmount"
    app:fadeOffset="4"
    app:initialRadius="19dp"
    app:makeWavesFade="true"
    app:waveColor="@color/defaultTextColor"
    app:waveSpaceStep="18dp"
    app:waveWidth="2"
    app:wavesAmount="12"
    app:wavesAreVisible="true" />
```

Рисунок 3.32 – Вигляд елементу WiFiWavesView у макеті головного екрану

Основні параметри для редагування (рис. 3.33, рис. 3.34):

- drawBy – параметр, що встановлює алгоритм заповнення вільного місця хвилями. Доступно два варіанти: allowedSpace – додавати хвилі залежно

від розмірів макету, та wavesAmount – створення виключно зазначеної кількості хвиль;

- makeWavesFade – параметр, що регулює прозорість хвиль залежачи від їх розташування відносно центру; приймає значення true або false ;
- waveColor – параметр, що описує колір хвиль;
- waveSpaceStep – параметр, що встановлює дистанцію між хвилями;
- waveWidth – параметр, що регулює розмір хвилі;
- wavesAmount – параметр, що регулює кількістю хвиль. Цей параметр ігнорується, якщо drawBy має значення allowedSpace;
- wavesAreVisible – флаг видимості хвиль.

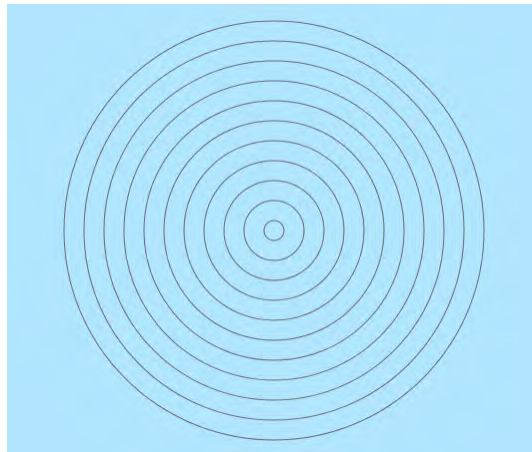


Рисунок 3.33 – Статичний вигляд хвиль із встановленим параметром makeWavesFade =«false»

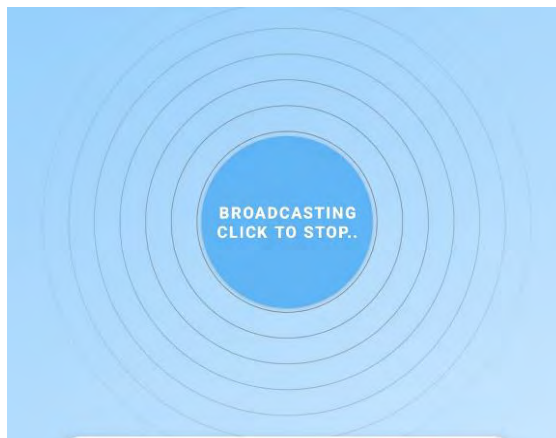


Рисунок 3.34 – Скриншот анімованих хвиль у застосунку

3.3.3 Екран завантаження застосунку (SplashFragment)

На сьогодні, згідно з порадами оформлення застосунку, використовувати екран завантаження, або, як його ще називають, SplashScreen вважається марнуванням часу користувача, бо застосунок вже міг би працювати деякий час [17]. Не зважаючи на це було вирішено додати такий екран, а також анімувати його елементи.

Ознайомитись наглядно з цим ви можете у самому додатку. Запуск та робота анімації відбувається у фрагменті SplashFragment. Його робота – це відображення елементів, що схожі за формою на текстові повідомлення, що за певний час переміщуються на початкові позиції із прозорого стану до стану повної видимості. Результат анімації зображено на рисунку 3.35.

Як тільки остання анімація завершує свою роботу – викликається метод `notifySplashAnimationFinished()`, що оновлює певний стан додатку, та повідомляє, що анімація завершилась і перехід до головного екрану дозволено. За увесь цей час система встигає створити усі необхідні дані та ініціалізувати об'єкти для роботи застосунку.

Метод `notifySplashAnimationFinished()` оголошений лише у класах `LaunchActivity` та `LaunchFragment`. `MainActivity` наслідується від `LaunchActivity`, а `SplashFragment` від `LaunchFragment`.

Реалізація методу в `LaunchFragment`.

```
fun notifySplashAnimationFinished(){
    if (isAdded) { // Перевіряє, чи додано фрагмент до активіті
        with(requireActivity() as? LaunchActivity) {
            // перевірка чи являється поточне активіті LaunchActivity
            this?.notifySplashAnimationFinished()
            // Виклик методу notifySplashAnimationFinished у класі
            LaunchActivity
        }
    }
}
```

Реалізація методу в LaunchActivity.

```
protected val splashAnimationFinished = MutableSharedFlow<Boolean>()

fun notifySplashAnimationFinished(){
    lifecycleScope.launch {
        splashAnimationFinished.emit(true)
    }
}
```

Тим часом до об'єкту `splashAnimationFinished` додається підписник у `MainActivity`, що реагує на зміни. Як тільки значення оновлюється на «true», то йде перевірка на поточний статус з'єднання із сервером. Якщо статус є типом «`ConnectivityStatus.Connected.Authenticated`», то спрацьовує перехід на вікно із переліком доступних чатів, інакше користувач буде перенаправлений до головного вікна. Реалізація такого алгоритму.

```
splashAnimationFinished.filter { it == true }.collect {
    withContext(Dispatchers.Main){ // перехід на головний потік
        navController?.let {
            if (it.currentDestination?.id == R.id.splashFragment){
                if (vm.connectivityStatus.value.second is
ConnectivityStatus.Connected.Authenticated){

it.navigate(R.id.action_splashFragment_to_chatListFragment_NO_FILTER)
                } else {
                    it.navigate(R.id.action_splashFragment_to_authFragment)
                }
            }
        }
    }
}
```

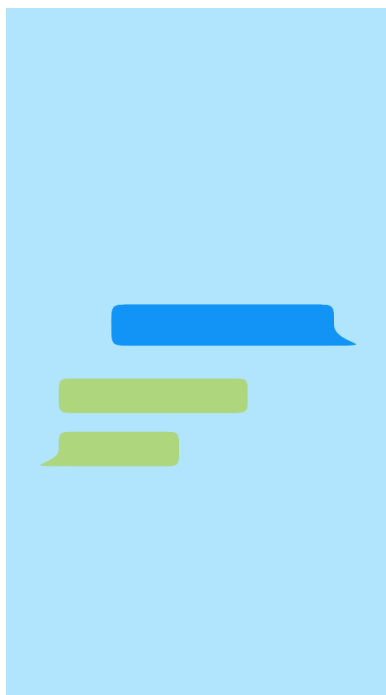


Рисунок 3.35 – Экран завантаження додатку

3.3.4 Головний екран застосунку (AuthFragment)

Вигляд AuthFragment (рис. 3.36).

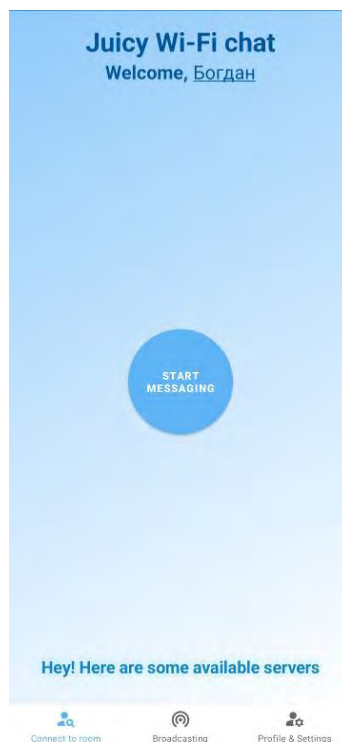


Рисунок 3.36 – Загальний вигляд AuthFragment

Головний екран складається з трьох головних елементів. Зверху розташовано текст-привітання та ім'я користувача. Ім'я користувача виділено нижнім підкресленням та виступає кнопкою на екрані (рис. 3.37).



Рисунок 3.37 – Вигляд тексту-привітання із відображенням імені користувача

При натисканні на виділений текст користувачу відкривається вікно редагування імені. Це зручно та допомагає швидше ознайомитись із функціоналом додатку. Перехід виконується завдяки компоненту навігації розглянутому у пункті 3.1.5 та команди `navigate(R.id.action_authFragment_to_profileSettingsFragment)`, де `action_authFragment_to_profileSettingsFragment` це ідентифікатор шляху, що визначений у файлі навігації та вміщує інформацію про анімацію переходів.

Далі, під текстом привітання знаходиться кнопка із заголовком «Start messaging». При натисканні запускається `ServerFinderService`, а текст кнопки змінюється на «Stop», що пропонує користувачу зупинити пошук (рис. 3.38). Цей сервіс допомагає знаходити сервери поблизу, створені та налагоджені додатком Juicy Wi-Fi Chat. Результатом роботи сервісу є дані, що описують кожний знайдений сервер, його адресу, ім'я, ідентифікатор та кількість підключених користувачів у даний час. Окрім цього було написано власний функціонал автоматичної зупинки сервісу пошуку за відліком певного часу, а саме – однієї хвилини (рис. 3.39).

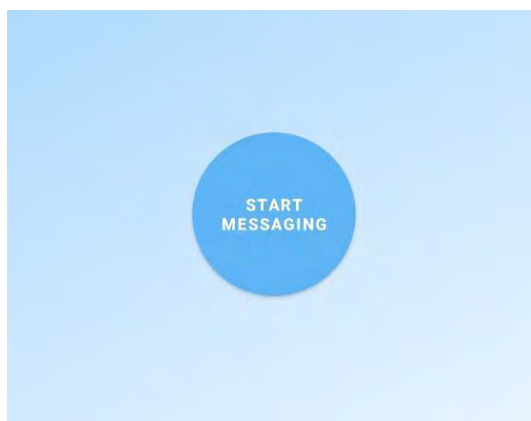


Рисунок 3.38 – Вигляд кнопки, що починає роботу сервісу пошуку серверів поблизу

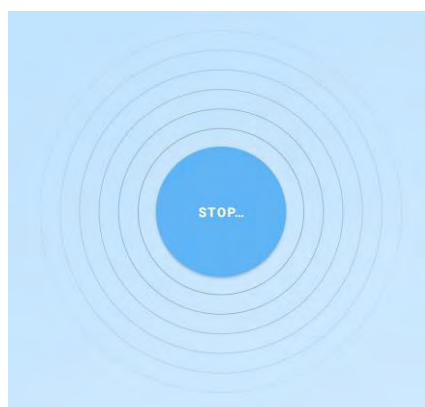


Рисунок 3.39 – Вигляд кнопки, що зупиняє роботу сервісу пошуку серверів поблизу на головному фрагменті

Як тільки сервіс починає роботу, то над нижнім меню з'являється текст-повідомлення, що інформує користувача про початок сканування. Як тільки дані знайдених серверів оновлюються – текст змінюється та інформує про це. Переглянути такі повідомлення можна за рисунком 3.40.

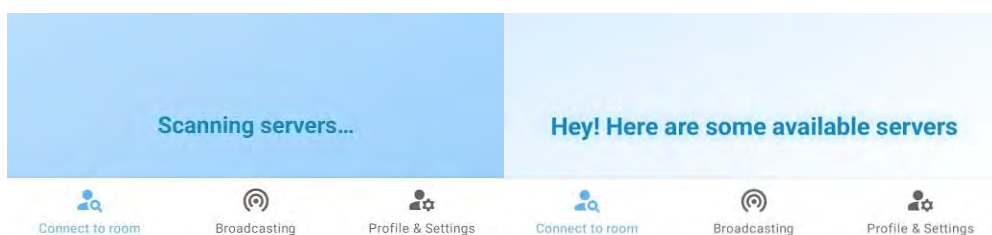


Рисунок 3.40 – Тексти, що інформують користувача про стан сканування серверів

Також знизу розташовано головне меню, що є класом `BottomNavigationView` та надає змогу синхронізувати себе із `NavigationUI` компонентом. Це меню дає змогу пересуватись між трьома важливими вкладками – пошук кімнат, управління режимом хоста та перегляд профілю користувача.

Важливо зазначити, що меню як компонент розташований на макеті головного активіті – `MainActivity`, а не на `AuthFragment`. Таким підхід дозволяє уникнути повторного використання коду і робить меню доступним на всіх екранах окрім екрану діалогу з користувачем, там воно програмно скривається.

3.3.5 Редагування користувацького профілю та серверу

Форми редагування профілю та особистого серверу схожі між собою як за виглядом, так і за функціоналом.

3.3.5.1 Огляд форми профіля (`ProfileSettingsFragment`)

Фрагмент `ProfileSettingsFragment` відповідає за перегляд і редагування профілю. На поточний час підтримується тільки зміна імені. Фрагмент має свою особисту вьюмодель - `UserSettingsViewModel`, що відповідає за уся логіку із збереженням та порівнянням даних користувача(див. функціонал класу на рис. 3.41).

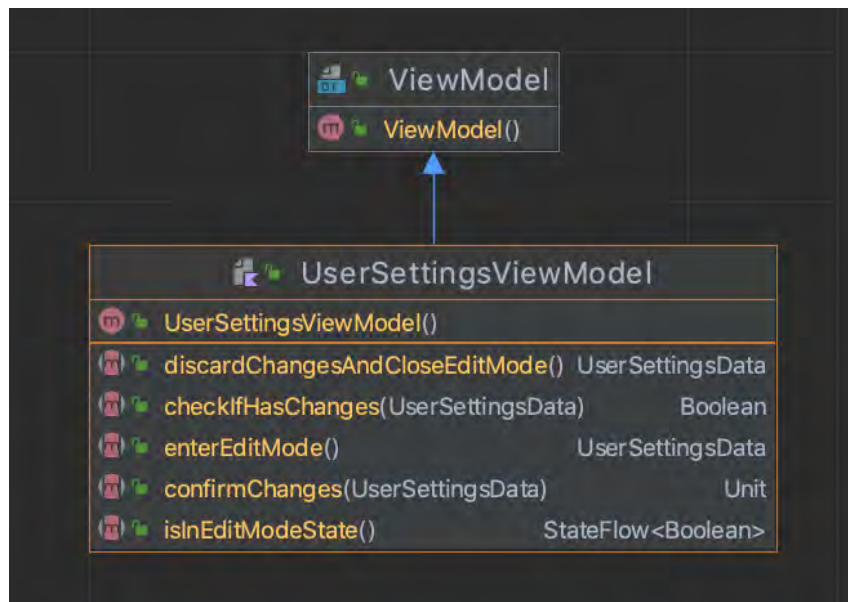


Рисунок 3.41 – Діаграма абстрактного класу UserSettingsViewModel та перелік методів

Макет фрагменту складається з кнопки налаштувань у верхньому правому кутку, та макету профілю. Макет профіля вміщую в собі кнопку, що перемикає статус «isInEditMode» (режим редагування). На основі цього статусу оновлюються інші елементи: якщо користувач зайшов у режим редагування, то кнопка «Edit» змінює свій текст на «Cancel», «Save changes» та поле «User name» стає активним і користувач може редагувати його.

Вхід до режиму редагування відбувається за допомогою виклику методу UserSettingsViewModel::enterEditMode(), що повертає копію збережених даних про користувача. У цей час можна редагувати дані допоки користувач не вийде з цього режиму. Було додано перевірки на валідність даних, а саме – поле імені не повинно бути порожнім. У разі якщо воно порожнє, то встановлюється ім'я пристрою користувача. Як тільки користувач вирішив закінчити редагування, то він може натиснути кнопку «Cancel» для відміни, або «Save changes» для збереження модифікацій.

Також було розроблено алгоритм зміни кольору профілю, що залежить від введеного імені. Таким чином можна розрізняти користувачів із списку

чатів значно швидше. Алгоритм, що генерує колір, на основі вхідних даних (в нашому випадку – ім'я користувача).

```
fun generateStrColorFromInput(input: String?): String {
    if (input == null || input == "") return "#000000"
    val hash = input.hashCode().toString().replace("-", "").toInt()
    val color = String.format("#%06X", hash)
    return color.substring(0, 7)
}
```

За замовчуванням, якщо користувач натисне на кнопку «Cancel» то буде показано вікно підтвердження дії (рис. 3.42).

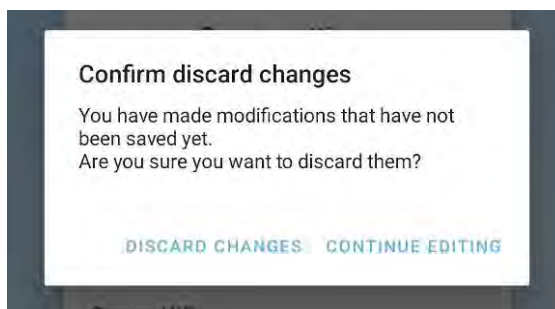


Рисунок 3.42 – Вікно-підтвердження відміни модифікацій

Вигляд фрагменту редагування профілю на рис. 3.43



Рисунок 3.43 – Загальний вигляд фрагменту редагування профілю

3.3.5.2 Огляд екрану конфігурацій серверу(HostSettingsFragment)

Як було зазначено – макети фрагменту профілю і фрагменту конфігурацій серверу схожі між собою. Макет складається із кнопки запуску або завершення роботи режиму хоста. Цей режим дозволяє створювати так звану кімнату, до якої можуть підключитись інші користувачі у радіусі роботи Wi-Fi. При натисканні на кнопку із текстом «Start broadcasting» запускається сервіс BroadcastingService, а текст кнопки змінюється залежно від поточного стану. Докладніше про цей сервіс див. пункт 3.2.5.2.

На фоні за кнопкою використовується кастомний елемент інтерфейсу WiFiWavesView, що візуально символізує роботу серверу.

Статус роботи передбачає 4 стани (рис. 3.44): «Disabled», «Preparing», «Broadcasting», «ClosedWithError».

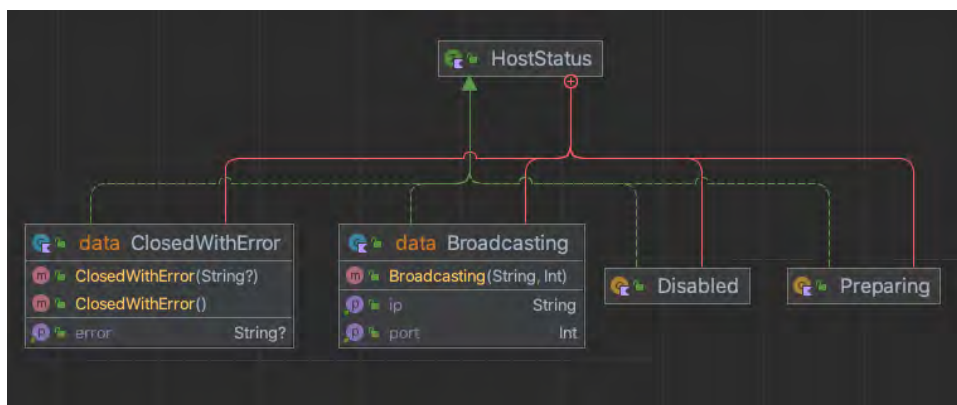


Рисунок 3.44 – Діаграма, що відображає визначені стани роботи сервісу BroadcastingService

Нижче від кнопки включення/вимкнення режиму хоста відображається форма редагування назви серверу та його ідентифікатору. Режим редагування працює так само, як було зазначено в пункті 3.3.5.1. Було додано перевірки на статус роботи серверу та входу користувача у режим редагування. Головне правило – користувач не повинен мати можливості редагувати профіль поки статус роботи серверу має значення «HostStatus.Broadcasting». Також

користувач не може включити сервер доки не вийде із режиму редагування. Для цього створено відповідні діалогові вікна (рис. 3.45).

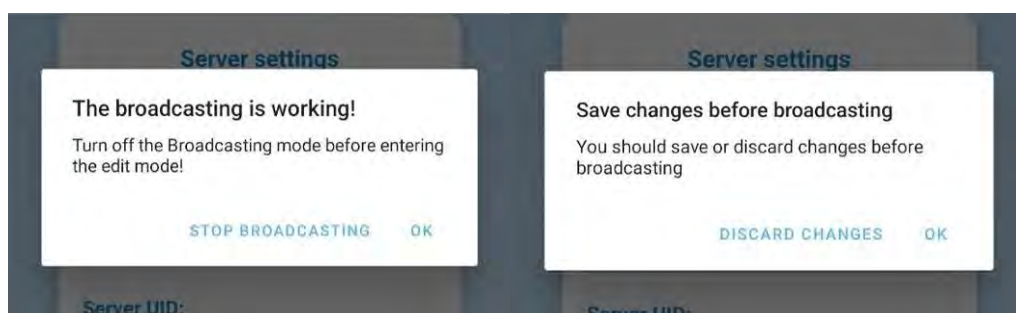


Рисунок 3.45 – Попереджувальні діалогові вікна

Кнопка «Revoke» справа від текстового поля ідентифікатору використовується задля очищення даних серверу. Таким чином видаляється уся інформація що використовувалась для створення кімнати та ідентифікації користувачів, а користувачі, що до цього моменту були зареєстровані у сервері за таким ідентифікатором, більше не зможуть отримати доступ до історії спілкування (рис. 3.46).

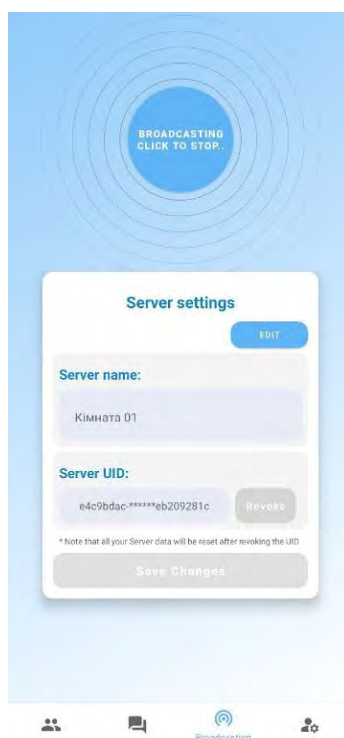


Рисунок 3.46 – Загальний вигляд фрагменту конфігурації серверу

3.3.6 Перегляд знайдених серверів/кімнат (FoundServersListFragment)

Функціонал фрагменту FoundServersListFragment дуже обмежений (рис. 3.47). Його ціль – відображення знайдених серверів поблизу. Фрагмент Макет фрагменту складається із списку елементів, що візуально описують модель ServerParams (рис. 3.48). За відображення даних відповідає клас AvailableServersAdapter, що наслідується від ListAdapter<ServerParamsWithSize, AvailableServersViewHolder>(...). ServerParamsWithSize це модель, що описує інформацію про знайдену кімнату та наслідується від інтерфейсу ServerParams. Вона зберігає IP адресу серверу, назву, ідентифікатор та кількість підключених користувачів.

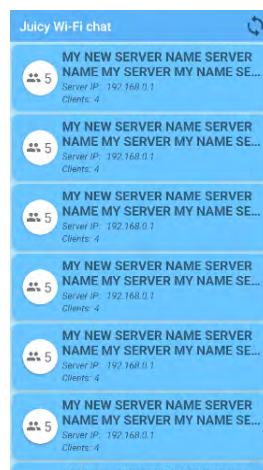


Рисунок 3.47 – Вигляд фрагменту FoundServersListFragment із тестовими даними

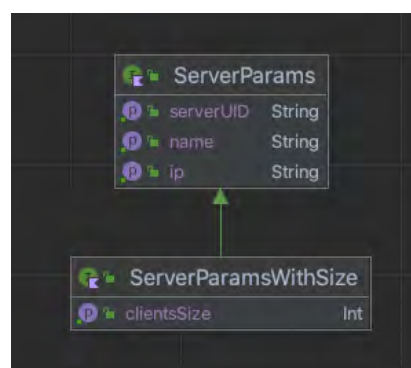


Рисунок 3.48 – Модель інтерфейсів ServerParams та ServerParamsWithSize

Після натискання на елемент списку – з’являється діалогове вікно, що запитує користувача про підтвердження дії (підключення до обраного серверу) (рис. 3.49).

Після вибору варіанту «Connect» («Приєднатися») зупиняє свою роботу сервіс що займався пошуком серверів. Зразу після цього статус підключення змінюється на «ConnectivityStatus.ChosenServerToConnect», і наступним кроком починає свою роботу інший сервіс – ConnectedToServerService за допомогою методу `startMyForegroundService<ConnectedToServerService>(requireContext())`. Детальна робота класу ConnectedToServerService описано у пункті 3.2.5.3.

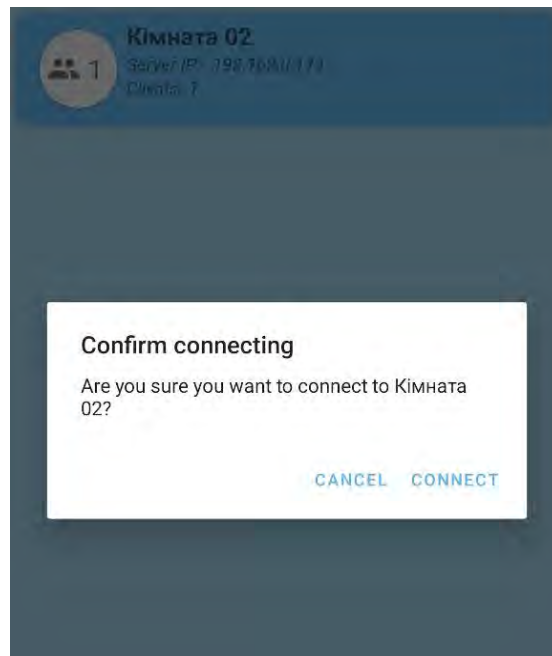


Рисунок 3.49 – Відображення вікна підтвердження щодо з’єднання до обраного серверу

3.3.7 Фрагмент, що відображає чати для спілкування (ChatListFragment)

Фрагмент ChatListFragment візуально складається із верхньої панелі, та списку типу RecyclerView, що відображає перелік чатів. За відображення елементів відповідає створений клас ChatAdapter, що наслідується від ListAdapter<Chat, ChatAdapter.ChatHolder>(..). Різниця між RecyclerView.Adapter<T> та ListAdapter<T, R> полягає в тому, що ListAdapter має розширений функціонал та може порівнювати нові елементи зі старими за допомогою класу DiffUtil.ItemCallback<T>, і якщо вони не змінюються, то пропускати оновлення. Це може значно оптимізувати роботу застосунку, бо відпадає потреба оновлювати увесь список елементів разом (рис. 3.50, рис. 3.51).



Рисунок 3.50 – Загальний вигляд фрагменту ChatListFragment із тестовими даними

Приклад порівняння елементів типу Chat у адаптері.

```
class ChatDifferenceCallback : DiffUtil.ItemCallback<Chat>() {
    override fun areContentsTheSame(oldItem: Chat, newItem: Chat): Boolean {
        // Порівняння на схожість елементів за контентом
        return oldItem.isOnline == newItem.isOnline &&
            oldItem.lastMessage.message == newItem.lastMessage.message &&
            oldItem.unreadMessagesCount == newItem.unreadMessagesCount &&
            oldItem.lastMessage.dateInMs == newItem.lastMessage.dateInMs
    }

    override fun areItemsTheSame(oldItem: Chat, newItem: Chat): Boolean {
        // Перевірка на еквівалентність одного елементу з іншим за
        ідентифікатором.
        return oldItem.userID == newItem.userID
    }
}
```

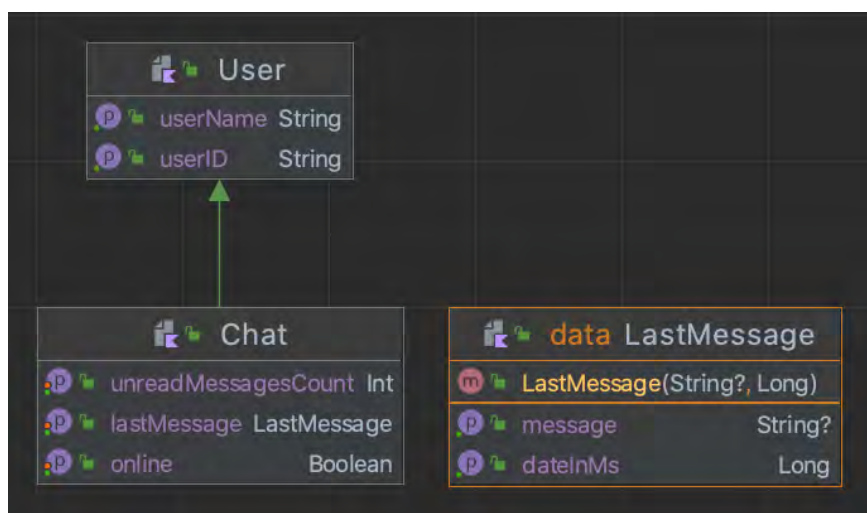


Рисунок 3.51 – Модель класу Chat, що наслідується від User, та клас даних LastMessage

Клас **ChatHolder** відповідає за наповнення шаблону (рис. 3.52) інформацією отриманої з об'єкту типу **Chat**. **ChatHolder** має лише один метод із назвою `bind`, та викликається тоді, коли елемент стає видимим на екрані, або при його оновленні. Цей метод наповнює шаблон наступними даними про чат:

- присвоюється ім'я чату заголовку шаблону;

- присвоюється текст останнього повідомлення. Якщо чат не має збережених даних щодо останнього повідомлення, то показується відповідний текст – «History is empty»;
- далі йде перевірка на час останнього відправленого повідомлення. Якщо дата виражена у Мілі секундах дорівнює 0, то дата не відображається;
- додатковим функціоналом є перегляд статусу «Онлайн» користувача. Цей статус зберігається у полі Chat::isOnline. Залежно від значення буде обрано колір кружечку, що відображається поруч з іменем користувача;
- також відображається лічильник непрочитаних повідомлень. Він розташований з правої сторони шаблону.

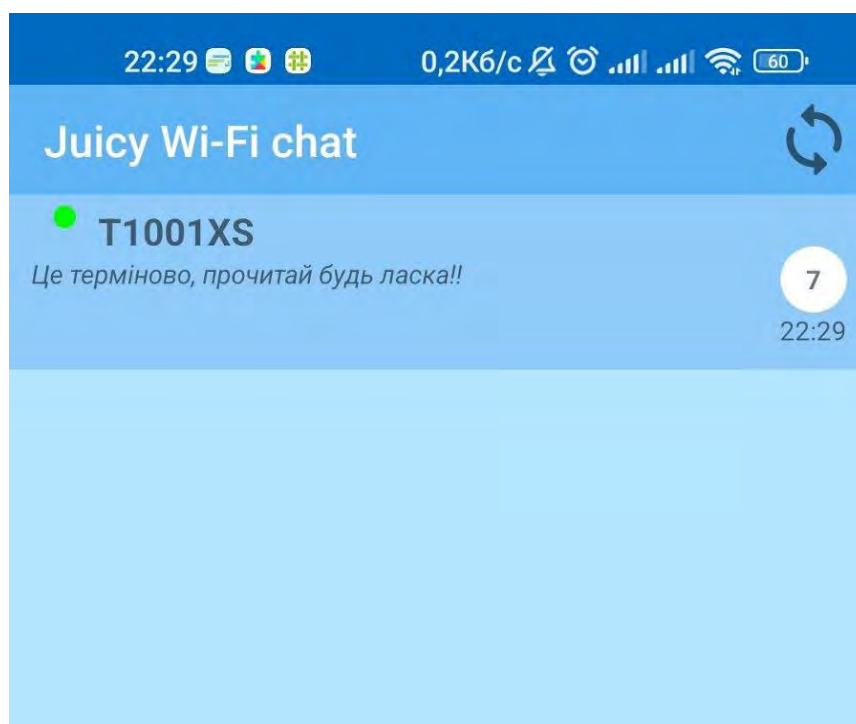


Рисунок 3.52 – Відображення шаблону із інформацією про чат

ChatListFragment зберігає свою вьюмодель, що відповідає за завантаження даних, що будуть відображатись у ChatListFragment та оновлення користувачів.

Як тільки користувач підключається та авторизується на сервері, то спрацьовує функціонал перетворення елементів меню. Замість вкладки

«Connect to room» з'являються дві інші: «All Chats» («Всі чати») та «Active Chats» («Активні чати»). Це реалізовано у MainActivity, у методі `updateBottomMenuAccordingToIsConnectedState(status: ConnectivityStatus)`.

```
private fun updateBottomMenuAccordingToIsConnectedState(status:
ConnectivityStatus) {
    if (navController?.currentBackStackEntry?.destination?.id ==
R.id.chatFragment){
        return
    }
    when {
        status is ConnectivityStatus.Connected.Authenticated -> {
            if (!binding.bottomMenu.isVisible binding.bottomMenu.menu.size !=
ROUTES_MENU_WITH_CHATS.size) {
                // Якщо поточне меню не є меню із чатами
                binding.bottomMenu.menu.clear()
                // Встановлення меню із елементами доступу до чатів
                binding.bottomMenu.inflateMenu(R.menu.menu_with_chats)
                hideShowBottomMenu(isVisible = true)
            }
        }
        // Перевірка чи являється переданий у аргументах тип статусу одним із
перелічених у listOf(..)
        status isOneOf listOf(
            ConnectivityStatus.ChosenServerToConnect::class.java,
            ConnectivityStatus.Connected.ConnectedToServer::class.java,
            NotConnected::class.java
        ) -> {
            if (!binding.bottomMenu.isVisible binding.bottomMenu.menu.size !=
ROUTES_GENERAL_MENU.size) {
                binding.bottomMenu.menu.clear()
                // Встановлення стандартного меню
                binding.bottomMenu.inflateMenu(R.menu.general_menu)
                hideShowBottomMenu(isVisible = true)
            }
        }
    }
}
```

У нових версіях застосунку окрім періодичного оновлення було додано кнопку оновлення користувачів власноруч. Це відбувається при натисканні на значок «Оновлення» у правому верхньому куточку. Код обробки натискання.

```
binding.imageUpdateChats.setOnClickListener {
    vm.loadUsers()
    Toast.makeText(context, R.string.toast_users_updated,
        Toast.LENGTH_SHORT).show()
}
```

3.3.8 Фрагмент для спілкування (ChatFragment)

Форма спілкування має мінімалістичний вигляд та складається із верхньої панелі – титульної, що відображає ім'я співрозмовника, безпосередньо елементу RecyclerView, що відображає історію спілкування та форми вводу текстового повідомлення.

Для обробки даних та дій було створено ChatViewModel клас, що зберігає усю логіку роботи із даними фрагменту(рис. 3.53). Вона дозволяє підписуватись на оновлення історії повідомлень й відправляти такі текстові повідомлення.

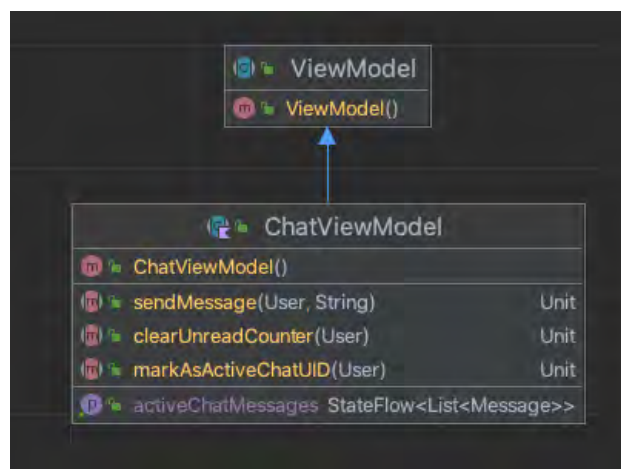


Рисунок 3.53 – Діаграма класу ChatViewModel

Для відображення текстових повідомлень було розроблено адаптер MessagesAdapter та створено два класи типу ViewHolder (рис. 3.54): MineMessagesHolder та NotMineMessagesHolder, що відрізняються спроектованим шаблоном що відображається. Для MineMessagesHolder використовується синій прямокутник із закругленими краями та текст часу відправлення прилягає до правого боку. Було створено альтернативний вигляд, що описує повідомлення співрозмовника, розділяючи елементи візуально за кольором. Це зелений прямокутний із такими ж закругленими краями, але дата відправлення прилягає до лівого боку. На рисунку 3.55 показано загальний вигляд форми ChatFragment.

Для розуміння, чи являється автором повідомлення співрозмовник було використано порівняння на ідентифікатор автору повідомлення. Таким чином регулюється відображення повідомлень. Код з MessagesAdapter, що вирішує який тип повідомлення відобразити.

```
override fun getItemViewType(position: Int): Int {
    if (getItem(position).senderId == userID)
        return VIEW_TYPE_MESSAGE_IS_NOT_MINE
    return VIEW_TYPE_MESSAGE_IS_MINE
}
```

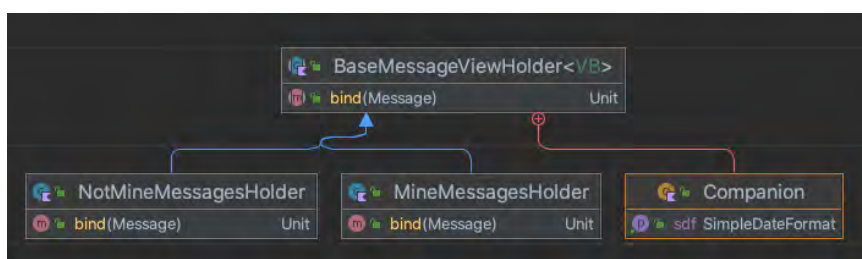


Рисунок 3.54 – Діаграма наслідування класів BaseMessageViewHolder класами

Оновлення історії повідомлень відбувається завдяки підписки на об'єкт «GlobalFlows.activeChatMessages», що зберігає перелік повідомлень активного чату. Активним чатом є чат, діалог якого відкрито у даний час. Якщо користувач покидає вікно діалогу, то значення «GlobalFlows.activeChatUID» змінюється на null, а разом із тим очищуються дані у змінній

«GlobalFlows.activeChatMessages». Оновлення даних при зміні активного чату відбувається у класі DataRepo наступним чином.

```

override fun setActiveChat(userID: UserID?){
    scope.launch(Dispatchers.Main) {
        GlobalFlows.activeChatUID.emit(userID)
        if (userID == null) {
            GlobalFlows.activeChat.emit(null)
            GlobalFlows.activeChatMessages.emit(emptyList())
        } else {
            clearUnreadCounter(userID)
            GlobalFlows.activeChatMessages.emit(getMessages(userID))
            GlobalFlows.activeChat.emit(getChat(userID))
        }
    }
}

```



Рисунок 3.55 – Загальний вигляд фрагменту діалогу із співрозмовником

3.3.9 Фрагмент налаштувань додатку(SettingsFragment)

До фрагменту налаштувань можна потрапити лише через фрагмент перегляду та редагування профілю натиснувши на значок налаштувань, що знаходиться у верхньому правому куточку. Макет фрагменту складається із кнопки повернення назад та зі списку RecyclerView, що наслідує RecyclerView.Adapter<SettingsDestinationViewHolder>(). Вікно складається з переліку полів, що розмежовані лінією, та особистим обробником кліків. У фрагменті SettingsFragment завчасно визначено два таких поля: очищення даних, та перегляд інформації про застосунок, його версія та інформація про розробника. Було створено модель «Destination», що має підкласи Callback та ContextCallback (рис. 3.56). Їх можна використовувати в залежності від того, чи потрібен об'єкт контексту у тілі функції зворотного виклику. Такий підхід може допомогти винести функціонал та список полів за межі фрагменту, та використовувати той контекст, що буде передаватись кожного разу при натисканні (рис. 3.57).

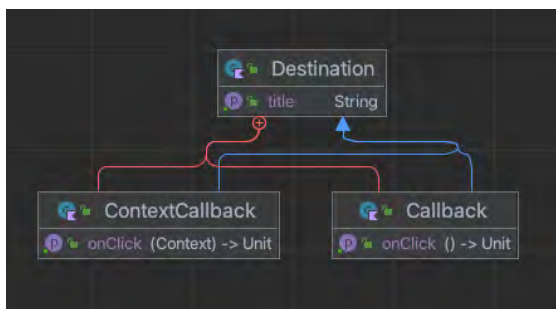


Рисунок 3.56 – Діаграма класу Destination та його підкласів

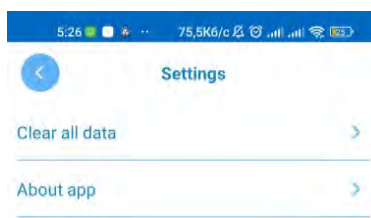


Рисунок 3.57 – Демонстрація фрагменту налаштувань

Після натискання на опцію очищення даних («Clear all data») користувачу відображається діалогове вікно, що інформує про наслідки такого рішення (рис. 3.58, рис. 3.59). Код, що буде виконаний при натисканні «Confirm» («Підтвердити»).

```
private fun invalidateAllData(){
    context?.let { context ->
        HostDataStorage.get(context).invalidateAllData()
        SessionStaticDataStorage.get(context).invalidateAllData()
        UserDataStorage.get(context).invalidateAllData()
        val prefsDir = File(context.dataDir, SHARED_PREFS_DIR)
        val allPrefs = if (prefsDir.exists() && prefsDir.isDirectory)
prefsDir.list() else emptyArray()
        val allSessionDynamicPrefs = allPrefs.filter {
            it.startsWith(SessionDynamicDataStorage.fileNamePrefix)
        }.toTypedArray()
        allSessionDynamicPrefs.forEach {
            context.deleteSharedPreferences(it)
        }
        Toast.makeText(context, R.string.all_data_cleared_successfully,
Toast.LENGTH_SHORT).show()
    }
}
```

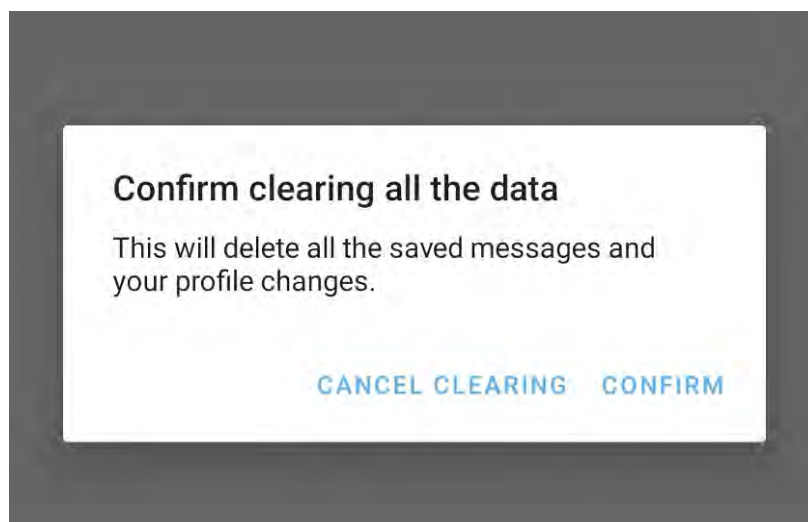


Рисунок 3.58 – Діалогове вікно із підтвердженням очищення даних

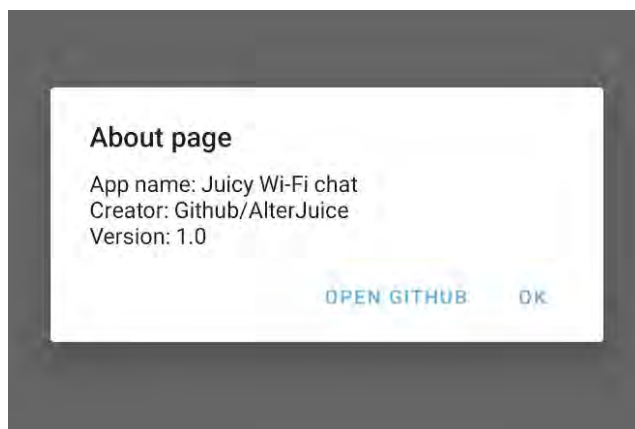


Рисунок 3.59 – Діалогове вікно з інформацією про розробника та версію застосунка

3.4 Огляд інтерактивних повідомлень

3.4.1 Текстові повідомлення від учасників мережевої кімнати

Так як функціонал додатку має функціонал типового чату, тож було реалізовано механізм генерування текстових повідомлень. Повідомлення з'являються як тільки користувач вийшов із додатку, або знаходиться у іншому активному чаті. Редагування відображення повідомлень відбувається у вікні налаштувань додатку в системі Android (рис. 3.60).

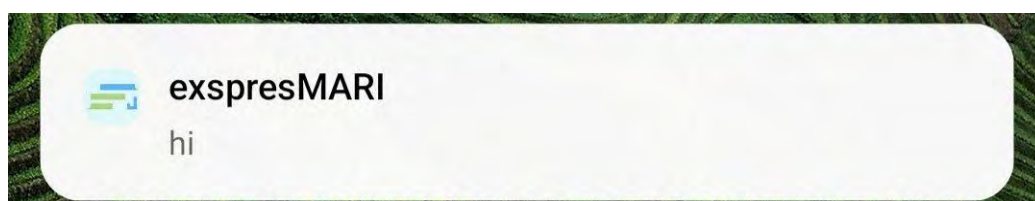


Рисунок 3.60 – Відображення вигляду текстового повідомлення від іншого користувача

Вигляд повідомлень є лаконічним: іконка додатку, ім'я користувача, що відправив повідомлення, та текст повідомлення. Із генеруванням вигляду повідомлення генерується й дія, що буде виконана при натисканні. За замовчуванням відкривається відповідне вікно діалогу між користувачами (рис. 3.61).

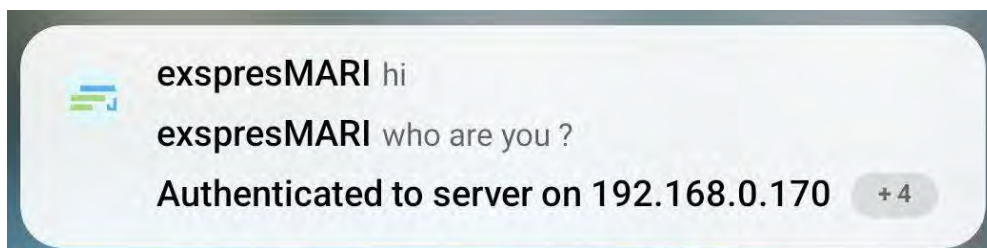


Рисунок 3.61 – Вигляд згрупованих повідомлень

3.4.2 Інформаційні повідомлення про статус пошуку доступних серверів/кімнат

Після того, як користувач натискає на центральну кнопку головного екрану – починається пошук кімнат поблизу. Задля того, щоб користувач розумів що пошук працює – було вирішено створити окреме повідомлення, що інформує користувача про наявність доступних кімнат та дві кнопки у межах повідомлення: Перегляд серверів (кімнат) та зупинити пошук. Повідомлення оновлюється кожного раз як тільки знаходить нову кімнату.

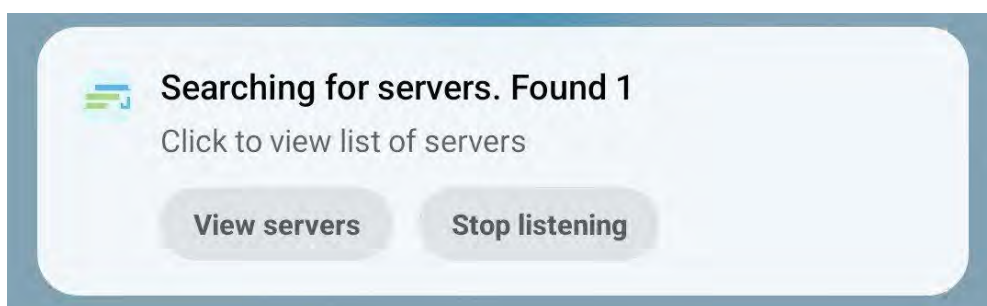


Рисунок 3.62 – Відображення повідомлення про пошук кімнат

При натисканні на кнопку перегляду кімнат – користувача буде переадресовано до фрагменту із списком доступних серверів (рис 3.62).

3.4.3 Інформаційні повідомлення про статус підключення

Було визначено 4 стани підключення користувача до серверу у додатку:

- «Не з'єднаний»;

- «Йде підключення»;
- «З'єднався з сервером»;
- «Користувач авторизован».

Як тільки статус змінюється, то викликається команда на зміну повідомлення у системі Android за певним ідентифікатором. Перелік доступних виглядів таких повідомлень відображено на рис. 3.63 – 3.66. Як тільки користувач з'єднався з сервером, то йому дається змога перейти до списку користувачів, що підключені до тієї ж кімнати завдяки кліку на повідомлення.

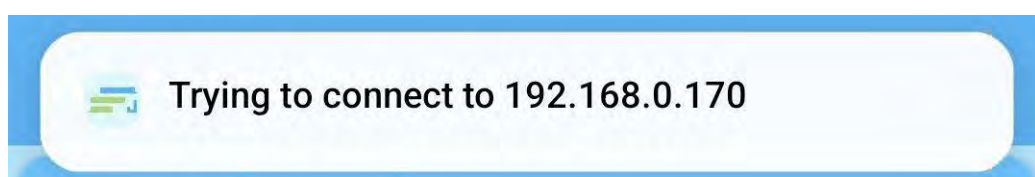


Рисунок 3.63 – Відображення статусу «Йде підключення»

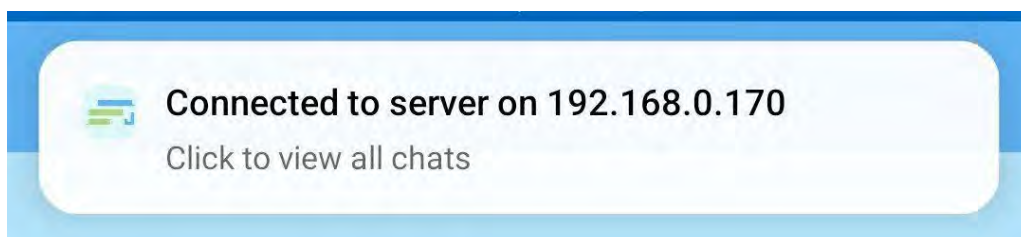


Рисунок 3.64 – Відображення статусу «З'єднався з сервером»

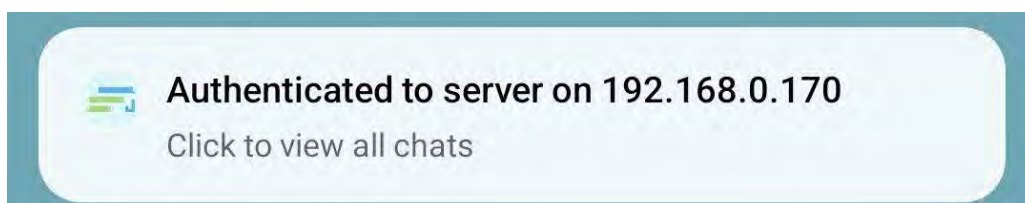


Рисунок 3.65 – Відображення статусу «Користувач авторизован»

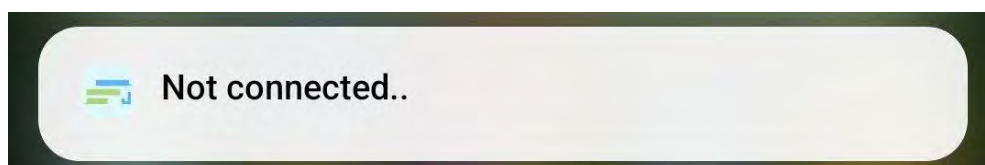


Рисунок 3.66 – Відбраження статусу «Не з'єднаний»

3.4.4 Інформаційні повідомлення про статус роботи серверу

Як тільки користувач включає режим хосту (Broadcasting), то показується відповідне повідомлення, що зберігає IP серверу (рис. 3.67). При натисканні на повідомлення користувача буде направлено до екрану редагування конфігурацій серверу. Таке повідомлення неможливо прибрати звичайним жестом видалення. Таким чином користувач завжди може дізнатися чи працює його сервер у даний момент.

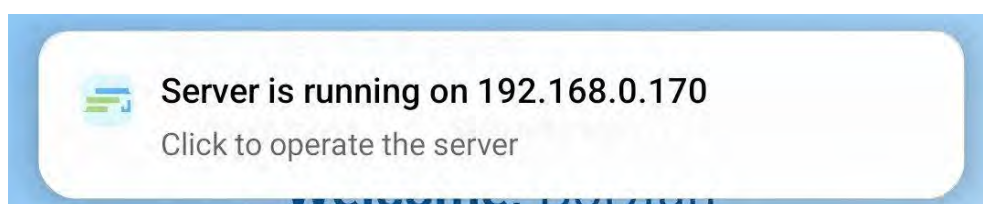


Рисунок 3.67 – Демонстрація вигляду повідомлення про статус роботи серверу

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУНКУ

У цьому розділі розглянуто процес роботи додатку, тестування на різних пристроях та версіях Android.

4.1 Процес роботи за стосунку

Користувач запускає застосунок нажимаючи на іконку «Juicy Wi-Fi Chat».

Система запускає процес ініціалізування компонентів, що потрібні для роботи додатку, та повертає вікно MainActivity.

Користувачу відображається екран завантаження.

Після закінчення анімації перевіряється статус підключення.

Якщо користувач з'єднано із сервером, то показується фрагмент із переліком доступних чатів, а меню змінюється на таке, що не має посилання на головне вікно.

У разі якщо статус з'єднання має значення відмінне від «Connected», то відкривається головний фрагмент, що дозволяє користувачу почати спілкування.

При переході на головний екран користувачу відображається меню із такими елементами: «З'єднатись із кімнатою», «Режим хоста», «Налаштування профілю».

У вікні «Налаштування профілю» користувач має змогу редагувати своє ім'я, що буде відображатись у переліку інших чатів.

Для створення особистого серверу, що буде служити мережевою кімнатою для користувачів, слід перейти до вікна «Broadcasting». Користувач може змінювати ім'я серверу, його ідентифікатор та статус роботи, що може бути або включеним або вимкнутим. Як тільки статус мережевої кімнати має значення «Broadcasting», що означає що сервер працює, інші користувачі можуть з'єднатись із вашим пристроєм.

Для початку пошуку доступних серверів користувач нажимає на центральну кнопку круглої форми із текстом «Start messaging» на головному екрані.

Після цього здійснюється пошук та з'являються інтерактивні повідомлення у панелі сповіщень. Таке повідомлення інформує користувача про нові знайдені мережеві кімнати створені іншим користувачем у додатку «Juicy Wi-Fi Chat».

Після переходу на вікно переліку доступних мережевих кімнат, користувач має змогу обрати одну з таких кімнат натиснувши на відповідне поле.

Користувач підтверджує з'єднання із обраною кімнатою.

Користувач переходить до вікна переліку доступних чатів для спілкування, а меню змінюється на таке, що має 4 вкладки: «Всі чати», «Активні чати», «Режим хоста», «Налаштування профілю».

Користувач має змогу переглядати чати, інформація про яких зчитується із сховища додатку, та відкривати їх шляхом кліку на поле чату.

Користувач потрапляє на вікно перегляду історії спілкування, а також має змогу набирати та відправляти повідомлення співрозмовнику.

Під час роботи застосунку користувач може отримувати повідомлення, та переглядати їх у панелі сповіщень. Вони є інтерактивними, тож при натисканні відкривається чат із співрозмовником, що відправив таке повідомлення.

Усі повідомлення зберігаються у пам'яті та записуються у файл.

Для виходу із кімнати користувач має натиснути на іконку виходу на верхній панелі вікна «Усі чати», або «Активні чати».

Користувач також має можливість очищення усіх збережених даних у вікні налаштувань натиснувши на іконку шестерні у вікні «Налаштування профілю».

4.2 Тестування за стосунку

Було проведено ряд тестувань серед декількох пристроїв із різними версіями Android. Результати занотовано у таблицю 4.1. Розглянуто такі властивості:

- плавність анімації та роботи застосунку;
- перевірка коректного завантаження застосунку;
- перевірка адаптивності інтерфейсу;
- перевірка повідомлень, їх відображений текст та відгук на натискання;
- правильність та вчасність обробки статусів з'єднання;
- коректність роботи сервісів, що встановлюють зв'язок із іншими пристроями;
- коректність збереження та зчитування даних.

Таблиця 4.1 – Результати проведених тестувань

Критерій тестування	Назва пристрою та версія Android		
	Honor 7A – 8.1	Планшет T1001XS – 8.0	Xiaomi Mi10T – 12.0
Плавність анімації та роботи застосунку	Іноді анімація підлагує	Відмінно	Відмінно
Перевірка коректного завантаження	Коректно	Коректно	Коректно
Правильність та вчасність обробки статусів з'єднання	Відмінно	Відмінно	Відмінно
Перевірка повідомлень	Коректно	Коректно	Коректно
Коректність роботи сервісів (зв'язку із сервером користувача)	Коректно	Коректно	Коректно
Адаптивність інтерфейсу	Місцями - ні	Так	Так

Тестування було пройдено успішно на всіх пристроях окрім Honor 7A. Через екран малого розміру та поганої продуктивності системи пристрою Honor

7А було виявлено проблему некоректного відображення профілю користувача. Усі інші операції, що передбачені функціоналом застосунку, виконувались коректно та без проблем.

4.3 Висновки за розділом 4

У цьому розділі було оглянуто процес роботи шляхом опису основних функцій застосунку. Було також проведено тестування мобільних пристроїв та планшету із різними версіями операційної системи. Виявлено проблемні моменти роботи продукту. В цілому застосунок виконує всі основні задачі та є стабільним для роботи.

ВИСНОВКИ

Із виконанням дипломного проєкту було розроблено програмне забезпечення для підтримки зв'язку між користувачами у діапазоні роботи модулю Wi-Fi.

Було проведено огляд існуючих аналогів, їх порівняння, виявлення недоліків та переваг між собою. Для розробки застосунку було обрано мову програмування виключно Kotlin, а основне середовище Android Studio, що є офіційним продуктом для створення застосунків під ОС Android, та допоміжне – IntelliJ IDEA, функціонал якої допомагав створювати діаграми-опис розроблених класів.

Під час розробки Android застосунку для дипломного проєкту було проаналізовано та обрано архітектурний підхід MVVM. Було розглянуто та імплементовано патерни проектування, що робить кодову базу продукту якіснішою та приємнішою для читання. Також було обрано найсучасніші бібліотеки для особистого освоєння та роботи застосунку. Всю історію створення та код проєкту можна подивитись у репозиторію на GitHub: https://github.com/AlterJuice/AlterJuiceWiFiChat/tree/without_runtime_adapter

Було витрачено чимало часу на опанування документацій та порад щодо розробки продукту. Цей аналіз допоміг створити стабільну архітектуру, що у майбутньому допоможе легко модернізувати та доповнювати застосунок новим функціоналом.

Створений продукт відповідає основній меті розробки, та вирішує усі вищезазначені проблеми, а саме – можливість спілкування з користувачами у місцях відсутності доступу до мережі Інтернет.

Думки щодо вдосконалення застосунку у майбутньому:

- міграція механізму збереження даних, втілення сучасних інструментів для роботи із базами даних;
- підтримка відправлення медіа файлів користувачам;
- підтримка групових чатів;

- підтримка вибору та відображення фотографії користувача у чатах;
- шифрування даних.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАННЯ

1. Packet Loss vs. Distance Using Various WiFi Bitrates [Electronic resource]. – Access mode:
<https://inet.omnetpp.org/docs/showcases/wireless/errorrate/doc/index.html>
2. How Far can a WiFi Signal Travel [Electronic resource]. – Access mode: <https://www.actiontec.com/wifihelp/how-far-can-wifi-signal-travel-2/>
3. «Briar» [Електронний ресурс]. – Режим доступу:
<https://play.google.com/store/apps/details?id=org.briarproject.briar.android&hl=ru&gl=US>
4. «Bridgefy» [Електронний ресурс]. – Режим доступу:
<https://play.google.com/store/apps/details?id=me.bridgefy.main&hl=ru&gl=US>
5. «Talkie» [Електронний ресурс]. – Режим доступу:
https://play.google.com/store/apps/details?id=com.remaller.android.wifitalkie_lite&hl=ru&gl=US
6. Android statistics 2022 [Electronic resource]. – Access mode:
<https://www.businessofapps.com/data/android-statistics>
7. Android version history [Electronic resource]. – Access mode:
https://en.wikipedia.org/wiki/Android_version_history
8. Kotlin programming language [Electronic resource]. – Access mode:
[https://en.wikipedia.org/wiki/Kotlin_\(programming_language\)](https://en.wikipedia.org/wiki/Kotlin_(programming_language))
9. Kotlin, компиляция в байткод и производительность(часть 1) [Электронный ресурс]. – Режим доступа:
<https://habr.com/ru/company/inforion/blog/330060/>
10. Документація Android Source Project [Electronic resource]. – Access mode:
<https://developer.android.com/studio/projects/add-app-module#CreatingAModule>
11. Документація Android Source Project [Electronic resource]. – Access mode: <https://developer.android.com/studio/projects/android-library.html>

12. Патерн проектування «Спостерігач» [Електронний ресурс]. – Режим доступу: <https://refactoring.guru/uk/design-patterns/observer>
13. Патерни проектування «Одинак» [Електронний ресурс]. – Режим доступу: <https://refactoring.guru/uk/design-patterns/singleton>
14. «How to create a Singleton in Kotlin?» [Electronic resource]. – Access mode: <https://blog.mindorks.com/how-to-create-a-singleton-class-in-kotlin>
15. Патерн проектування «Стан» [Електронний ресурс]. – Режим доступу: <https://refactoring.guru/uk/design-patterns/state>
16. Документація Android Source Project - NavigationUI Component [Electronic resource]. – Access mode: <https://developer.android.com/guide/navigation/navigation-ui>
17. «Splash screens are evil! Don't use them» [Electronic resource]. – Access mode: <https://cyrilmottier.com/2012/05/03/splash-screens-are-evil-dont-use-them/>

ДОДАТОК А
Технічне завдання

Вступ

Найменування продукту «Juicy Wi-Fi Chat». Застосунок призначений для налагодження зв'язку між пристроями шляхом створення мережових кімнат та залучання у такі кімнати для підтримки комунікації із користувачами у межах стабільної роботи модулю Wi-Fi.

А.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка офлайн чат-застосунку із використанням вебсокетів», затверджене наказом Національного університету «Запорізька політехніка» № 102 від 27 квітня 2022 р.

А.2 Призначення розробки

Метою розробки програмного забезпечення є надання можливості користувачам створювати мережові групи, підключатися до таких груп та підтримувати комунікацію зі всіма учасниками групи у діапазоні роботи модулю Wi-Fi.

А.3 Вимоги до програми чи програмного вибору

А.3.1 Вимоги до функціональних характеристик

- Створення мережових кімнат;
- пошук мережових кімнат із користувачами поблизу, можливість з'єднання із такою кімнатою;
- можливість спілкування за допомогою обміну текстових повідомлень із всіма учасниками кімнати;
- можливість перегляду історії спілкування;

- можливість редагування даних кімнати (серверу) та профілю.

A.3.2 Вимоги до надійності

Вимоги до надійності роботи продукту:

- стабільна робота;
- обробка помилок для запобігання несанкціонованого закриття програми.

A.3.3 Умови експлуатації

Умови експлуатації співпадають з умовами експлуатації девайсу, на якому відбувається функціонування застосунку.

A.3.4 Вимоги до складу й параметрів технічних засобів

Технічні вимоги до обладнання:

- наявність екрану;
- наявність модулю Wi-Fi;
- не менше 1 ГБ ОЗУ;
- вільний простір не менш 50 МБ.

A.3.5 Вимоги до інформаційної й програмної сумісності

Програмні вимоги:

- сервіс віртуальної клавіатури;
- версія Android від 8.0.

A.3.6 Вимоги до маркування й упакування

Програма повинна мати маркування імені застосунку – «Juicy Wi-Fi Chat». У повному обсязі може бути записаний на компакт диск не менше 20МБ.

A.3.7 Вимоги до транспортування й збереження

Вимоги до транспортування і зберігання аналогічні вимогам, що пред'являються до лазерних дисків або карт пам'яті.

A.4 Вимоги до програмної документації

Програмна документація повинна включати: пояснювальну записку на розроблений продукт, вихідний код програми.

A.5 Техніко-економічні показники

Монетизація застосунку передбачає опублікування у Google Play маркет.

A.6 Порядок контролю й приймання

Передбачено захист дипломного проєкту у встановлені керівником проєкту строки.

ДОДАТОК Б
Текст програми

```

package com.edu.alterjuicechat.ui

import android.os.Bundle
import android.view.*
import com.edu.alterjuicechat.databinding.FragmentSplashBinding
import com.edu.alterjuicechat.extensions.animateExtended
import com.edu.alterjuicechat.ui.base.LaunchFragment

class SplashFragment :
LaunchFragment<FragmentSplashBinding>(FragmentSplashBinding::inflate) {
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }

    override fun onStart() {
        super.onStart()
        binding.reply1.slideAndFadeReplyDrawable(200f, 0)
        binding.reply2.slideAndFadeReplyDrawable(-200f, 1)
        binding.reply3.slideAndFadeReplyDrawable(-200f, 2) {
            notifySplashAnimationFinished()
        }
    }

    private fun View.slideAndFadeReplyDrawable(
        startPosition: Float, index: Int, endAction: () -> Unit = {}
    ) {
        this.animateExtended(
            slideFromX = startPosition, slideToX = 0f,
            fromScaleX = 1.3f, fromScaleY = 1.3f,
            toScaleX = 1f, toScaleY = 1f,
            fromAlpha = 0f, toAlpha = 1f,
            startDelay = index * 300L,
            duration = 500L,
            withStartAction = { this.visibility = View.VISIBLE },
            withEndAction = endAction
        )
    }
}

```

```
}
```

```
package com.edu.alterjuicechat.ui
```

```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.lifecycle.LifecycleScope
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.utils.generateConfirmationAlertDialog
import com.edu.alterjuicechat.databinding.FragmentHostSettingsBinding
import com.edu.alterjuicechat.domain.ServerName
import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.domain.model.HostStatus
import com.edu.alterjuicechat.domain.storage.server_data.HostDataStateNonNull
import com.edu.alterjuicechat.extensions.*
import com.edu.alterjuicechat.services.BroadcastingService
import com.edu.alterjuicechat.ui.base.BaseFragment
import com.edu.alterjuicechat.utils.generateAlertDialog
import com.edu.alterjuicechat.viewmodels.HostSettingsViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.collect
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import org.koin.androidx.viewmodel.ext.android.viewModel
import kotlin.random.Random
```

```
class HostSettingsFragment :
    BaseFragment<FragmentHostSettingsBinding>(FragmentHostSettingsBinding::inflate)
{
    private val vm by viewModel<HostSettingsViewModel>()

    private var dataWithChanges = HostDataStateNonNull("", "")

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }
}
```

```

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.root.animateBackgroundGradient()
    attachObservers()

    binding.settingsServerUIDChange.setOnClickListener {
        dataWithChanges = dataWithChanges.copy(serverUID =
vm.generateNewUID())
        updateServerSettingsInfoView(
            serverUID = dataWithChanges.serverUID
        )
    }
    binding.settingsServerSaveChanges.setOnClickListener {
        dataWithChanges = dataWithChanges.copy(
            serverName = binding.settingsServerNameInput.text.toString()
        )
        vm.confirmChanges(dataWithChanges)
    }
    if (vm.hostStatus.value is HostStatus.Broadcasting){
        makeStartBroadcastingButtonAvailableToStopService()
    } else if (vm.hostStatus.value is HostStatus.Disabled) {
        makeStartBroadcastingButtonAvailableToStartService()
    }
    binding.serverSettingsEditOrCancelButton.setOnClickListener {
        if (vm.isInEditModeState.value){
            dataWithChanges = dataWithChanges.copy(
                serverName = binding.settingsServerNameInput.text.toString()
            )
            if (vm.checkIfHasChanges(dataWithChanges)){
                // hasChanges
                generateConfirmationAlertDialog(
                    onDiscardChanges = {
                        dataWithChanges =
vm.discardChangesAndCloseEditMode()
                        updateServerSettingsInfoView(
                            serverUID = dataWithChanges.serverUID,
                            serverName = dataWithChanges.serverName
                        )
                    }
                ).show()
            } else {

```

```

        dataWithChanges = vm.discardChangesAndCloseEditMode()
        updateServerSettingsInfoView(
            serverUID = dataWithChanges.serverUID,
            serverName = dataWithChanges.serverName
        )
    }
} else {
    if (vm.hostStatus.value.isPreparingOnBroadcasting()){
        generateAlertDialog(context,
            getString(R.string.the_broadcasting_is_working),

getString(R.string.turn_off_broadcasting_before_entering_edit_mode),
            positiveButtonText = getString(R.string.text_ok),
            onPositiveButtonClick = {},
            negativeButtonText =
getString(R.string.stop_broadcasting),
            onNegativeButtonClick = {
                if
(vm.hostStatus.value.isPreparingOnBroadcasting()){
                    binding.broadcastingOnOfButton.performClick()
                }
            }
        ).show()
        return@setOnClickListener
    }
    dataWithChanges = vm.enterEditMode()
    updateServerSettingsInfoView(
        serverUID = dataWithChanges.serverUID,
        serverName = dataWithChanges.serverName
    )
}
}
updateWavesVisibility(vm.hostStatus.value is HostStatus.Broadcasting)
}

private fun updateWavesVisibility(isVisible: Boolean){
    binding.wifiWaves.showHideWaves(isVisible)
}

private fun updateServerSettingsInfoView(
    serverUID: ServerUID? = null,
    serverName: ServerName? = null
) {

```

```

        if (serverUID != null){
            binding.settingsServerUID.text = serverUID
        }
        if (serverName != null){
            binding.settingsServerNameInput.setText(serverName)
        }
    }

    private fun attachObservers(){
        lifecycleScope.launch {
            vm.isInEditModeState.collect { isInEditMode ->
                val multiplier = if (isInEditMode) 1 else 0
                withContext(Dispatchers.Main) {
                    binding.settingsServerSaveChanges.isEnabled = isInEditMode
                    binding.settingsServerUIDChange.isEnabled = isInEditMode
                    binding.settingsServerNameInput.isEnabled = isInEditMode
                    binding.settingsServerNameLayout.elevation = 10f *
multiplier

                    if (isInEditMode) {
                        binding.serverSettingsEditOrCancelButton.text =
getString(R.string.text_cancel)
                    } else {
                        binding.serverSettingsEditOrCancelButton.text =
getString(R.string.text_edit)
                    }
                }
            }
        }
        lifecycleScope.launch {
            vm.hostStatus.collect { status ->
                withContext(Dispatchers.Main) {
                    when (status) {
                        is HostStatus.Disabled -> {
                            updateWavesVisibility(false)
                            delay(getRandomTimeToDelayOnOfScanner())
                            makeStartBroadcastingButtonAvailableToStartService()
                        }
                        is HostStatus.Preparing -> {
                            binding.broadcastingOnOfButton.apply {
                                isEnabled = false
                                text =
getString(R.string.status_broadcasting_preparing)

```



```

        }
    }
    ).show()
    return@makeButtonAvailableToManageService
}
startMyForegroundService<BroadcastingService>(context)
}
}

private fun makeStartBroadcastingButtonAvailableToStopService(){

binding.broadcastingOnOfButton.makeButtonAvailableToManageService(getString(R.st
ring.status_broadcasting_click_to_stop)){
    stopMyForegroundService<BroadcastingService>(requireContext())
}
}

private fun getRandomTimeToDelayOnOfScanner(): Long {
    return Random.nextLong(800, 1500)
}
}

package com.edu.alterjuicechat.ui

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.core.os.bundleOf
import androidx.lifecycle.LifecycleScope
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.SimpleItemAnimator
import com.alterjuice.utils.RenewableJob
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.databinding.FragmentChatsListBinding
import com.edu.alterjuicechat.domain.NavArgKeys
import com.edu.alterjuicechat.domain.model.Chat
import com.edu.alterjuicechat.domain.model.User
import com.edu.alterjuicechat.extensions.isVisibleOrGone
import com.edu.alterjuicechat.extensions.makeVisible
import com.edu.alterjuicechat.ui.adapters.ChatAdapter
import com.edu.alterjuicechat.ui.base.BaseFragment

```

```

import com.edu.alterjuicechat.ui.model.ChatsFilter
import com.edu.alterjuicechat.viewmodels.ChatListViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.collect
import kotlinx.coroutines.flow.transform
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import org.koin.androidx.viewmodel.ext.android.sharedViewModel
import org.koin.core.component.KoinComponent

class ChatListFragment :
BaseFragment<FragmentChatsListBinding>(FragmentChatsListBinding::inflate),
KoinComponent {

    private val filterChatsBy by lazy {

navController?.currentBackStackEntry?.arguments?.getString(NavArgKeys.FILTER_CHA
TS_BY)?.let {
        ChatsFilter.valueOf(it)
    }?: ChatsFilter.NO_FILTER
    }

    private val vm by sharedViewModel<ChatListViewModel>()

    private val allChatsCollectorJob =
RenewableJob(::generateAllChatsCollectorJob)

    private fun filterChats(chats: List<Chat>, filter: ChatsFilter): List<Chat>
{
        return when (filter){
            ChatsFilter.NO_FILTER -> { chats }
            ChatsFilter.FILTER_ACTIVE -> { chats.filter{ it.isOnline } }
        }
    }

    private fun generateAllChatsCollectorJob() = lifecycleScope.launch {
        vm.allChats.transform{ allChats -> emit(filterChats(allChats,
filterChatsBy)) }.collect {
            withContext(Dispatchers.Main) {
                binding.textInfoAvailableChats.isVisibleOrGone(it.isEmpty())
                if (filterChatsBy == ChatsFilter.NO_FILTER){

```

```

        binding.textInfoAvailableChats.text =
getString(R.string.no_available_chats)
    } else if (filterChatsBy == ChatsFilter.FILTER_ACTIVE){
        binding.textInfoAvailableChats.text =
getString(R.string.no_available_active_chats)
    }
    chatsAdapter.submitList(it)
}
}

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    return binding.root
}

private val chatsAdapter by lazy { ChatAdapter(::onChatClick) }

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    with(binding.chatsList){
        adapter = chatsAdapter
        layoutManager = LinearLayoutManager(context)
        (itemAnimator as SimpleItemAnimator).supportsChangeAnimations =
false
    }
    binding.imageUpdateChats.setOnClickListener {
        vm.loadUsers()
        Toast.makeText(context, R.string.toast_users_updated,
Toast.LENGTH_SHORT).show()
    }
    attachObservers()
    (activity as? MainActivity)?.manipulateOnBottomMenu{
        this.makeVisible()
    }
}

override fun onResume() {
    super.onResume()
    vm.setActiveChatUID(null)
}

```

```

        private fun attachObservers(){
            allChatsCollectorJob.renew()
        }

        private fun onChatClick(user: User){
            vm.setActiveChatUID(user.userID)
            val args = bundleOf(
                NavArgKeys.CHAT_USER_ID to user.userID,
                NavArgKeys.CHAT_USER_NAME to user.userName,
            )
            when (filterChatsBy){
                ChatsFilter.NO_FILTER -> {

navController?.navigate(R.id.action_chatListFragment_NO_FILTER_to_chatFragment,
args)

                }
                ChatsFilter.FILTER_ACTIVE -> {

navController?.navigate(R.id.action_chatListFragment_FILTER_ACTIVE_to_chatFragment,
args)

                }
            }
        }
    }

package com.edu.alterjuicechat.ui

import android.content.Context
import android.net.wifi.WifiManager
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.lifecycle.LifecycleScope
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.databinding.FragmentAuthBinding
import com.edu.alterjuicechat.extensions.*
import com.edu.alterjuicechat.ui.base.BaseFragment
import com.edu.alterjuicechat.services.ServersFinderService
import com.edu.alterjuicechat.viewmodels.AuthViewModel
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.collect

```

```

import kotlinx.coroutines.flow.combine
import org.koin.androidx.viewmodel.ext.android.viewModel
import kotlin.random.Random

class AuthFragment :
BaseFragment<FragmentAuthBinding>(FragmentAuthBinding::inflate) {
    private val vm by viewModel<AuthViewModel>()

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.root.animateBackgroundGradient()

        binding.welcomeNameTitle.apply {
            this.makeTextUnderlined()
            text = vm.getSavedUsername()
            setOnClickListener {
                // go to user preferences
            }
        }

        navController?.navigate(R.id.action_authFragment_to_profileSettingsFragment)
    }

    binding.floatingScanningResults.setOnClickListener {

        navController?.navigate(R.id.action_authFragment_to_foundServersListFragment)
    }

    val mWifiManager =
requireContext().getSystemService(Context.WIFI_SERVICE) as WifiManager

    if (vm.serverScannerIsWorking.value){
        makeStartMessagingButtonAvailableToStopScanning()
    } else {
        makeStartMessagingButtonAvailableToStartScanning()
    }
}

```

```

        attachObservers()
    }

    private fun attachObservers(){
        lifecycleScope.launchWhenStarted {
            vm.serverScannerIsWorking.collect { isWorking ->
                withContext(Dispatchers.Main) {
                    if (isWorking) {
                        // Order may sense
                        delay(getRandomTimeToDelayOnOfScanner())
                        binding.scanningServersWaves.showWaves()
                        makeStartMessagingButtonAvailableToStopScanning()
                    } else {
                        // Order may sense
                        binding.scanningServersWaves.hideWaves()
                        delay(getRandomTimeToDelayOnOfScanner())
                        makeStartMessagingButtonAvailableToStartScanning()
                    }
                }
            }
        }
        lifecycleScope.launchWhenStarted {
            vm.availableServers.combine(vm.serverScannerIsWorking){ a, b -> a to
b }.collect {
                val (servers, isWorking) = it
                delay(getRandomTimeToDelayOnOfScanner())
                withContext(Dispatchers.Main){
                    if (servers.isNotEmpty()){
                        binding.floatingScanningResults.makeVisible()
                        binding.floatingScanningResults.text =
getString(R.string.scanning_status_found_servers)

binding.floatingScanningResults.animateWithScaleXYInfinitely()
                    } else {
                        if (isWorking){
                            binding.floatingScanningResults.makeVisible()
                            binding.floatingScanningResults.text =
getString(R.string.scanning_status_is_scanning)
                            binding.floatingScanningResults.animation?.cancel()
                        } else {
                            binding.floatingScanningResults.makeGone()
                        }
                    }
                }
            }
        }
    }

```

```

        }
    }
}

private fun makeStartMessagingButtonAvailableToStartScanning(){

binding.appStartMessaging.makeButtonAvailableToManageService(getString(R.string.
start_messaging)){
    startMyForegroundService<ServersFinderService>(requireContext())
}
}

private fun makeStartMessagingButtonAvailableToStopScanning(){

binding.appStartMessaging.makeButtonAvailableToManageService(getString(R.string.
stop_scanning)){
    stopMyForegroundService<ServersFinderService>(requireContext())
}
}

private fun getRandomTimeToDelayOnOfScanner(): Long {
    return Random.nextLong(800, 1500)
}
}

package com.edu.alterjuicechat.ui

import android.content.Context
import android.os.Bundle
import android.os.PowerManager
import android.widget.Toast
import androidx.constraintlayout.widget.ConstraintSet
import androidx.core.view.isVisible
import androidx.core.view.size
import androidx.lifecycle.LifecycleScope
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.extensions.viewBinding
import com.edu.alterjuicechat.databinding.ActivityMainBinding
import androidx.navigation.fragment.NavHostFragment
import androidx.navigation.ui.NavigationUI
import androidx.transition.TransitionManager
import com.alterjuice.utils.RenewableJob

```

```

import com.alterjuice.utils.isOneOf
import com.edu.alterjuicechat.domain.Consts
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import com.edu.alterjuicechat.domain.model.ConnectivityStatus.NotConnected
import
com.edu.alterjuicechat.domain.storage.session_data.static.SessionStaticData
import com.edu.alterjuicechat.domain.storage.user_data.UserData
import com.edu.alterjuicechat.extensions.isVisibleOrGone
import com.edu.alterjuicechat.navigation.data.ROUTES_GENERAL_MENU
import com.edu.alterjuicechat.navigation.data.ROUTES_MENU_WITH_CHATS
import
com.edu.alterjuicechat.navigation.data.navOptionsDefaultSingleTopForMenuItems
import com.edu.alterjuicechat.ui.base.LaunchActivity
import com.edu.alterjuicechat.viewmodels.MainActivityViewModel
import com.google.android.material.bottomnavigation.BottomNavigationView
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.*
import org.koin.android.ext.android.get
import org.koin.androidx.viewmodel.ext.android.viewModel

class MainActivity : LaunchActivity() {
    private val binding by viewBinding(ActivityMainBinding::inflate)

    private var jobCollectorIsConnected =
RenewableJob(::generateIsConnectedCollectorJob)
    private var jobCollectorSplashAnimationFinished =
RenewableJob(::generateSplashAnimationFinishedCollectorJob)

    private val vm by viewModel<MainActivityViewModel>()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)
        hideShowBottomMenu(false)
        binding.bottomMenu.menu.clear()
        navHost = NavHostFragment.create(R.navigation.nav_graph).also {
            supportFragmentManager
                .beginTransaction()
                .replace(R.id.mainFragmentContainer, it)
                .setPrimaryNavigationFragment(it)
        }
    }

```

```

        .commit()
    }
}

override fun onStart() {
    super.onStart()
    navController?.let {
        NavigationUI.setupWithNavController(binding.bottomMenu, it) }

        binding.bottomMenu.setOnItemSelectedListener {
            navController?.navigate(it.itemId, null,
            navOptionsDefaultSingleTopForMenuItems)
            true
        }
        val x = getSystemService(Context.POWER_SERVICE) as PowerManager
        val y = x.isIgnoringBatteryOptimizations(Consts.APP_PACKAGE)
        if (!y){
            //
            startActivity(Intent(ACTION_IGNORE_BATTERY_OPTIMIZATION_SETTINGS).also {
                //      it.setData(Uri.parse("packageName:
                "+"com.edu.alterjuicechat"))
                // })

            }
            //Toast.makeText(this, "isIgnoringBatteryOptimizations ${y}",
            Toast.LENGTH_LONG).show()
            attachObservers()
        }

        override fun onDestroy() {
            vm.stopAllJobs()
            cancelAllJobs()
            super.onDestroy()
        }

        fun manipulateOnBottomMenu(block: BottomNavigationView.() -> Unit){
            block.invoke(binding.bottomMenu)
        }

        private fun hideShowBottomMenu(isVisible: Boolean) {
            if (binding.bottomMenu.isVisible != isVisible) {
                binding.bottomMenu.isVisibleOrGone(isVisible)
                val constraints = ConstraintSet()

```

```

        constraints.clone(binding.root)
        constraints.clear(binding.mainFragmentContainer.id,
ConstraintSet.BOTTOM)
        if (isVisible) {
            constraints.connect(binding.mainFragmentContainer.id,
ConstraintSet.BOTTOM, binding.bottomMenu.id, ConstraintSet.TOP)
        } else {
            constraints.connect(binding.mainFragmentContainer.id,
ConstraintSet.BOTTOM, ConstraintSet.PARENT_ID, ConstraintSet.BOTTOM)
        }
        TransitionManager.beginDelayedTransition(binding.root)
        constraints.applyTo(binding.root)
    }
}

private fun cancelAllJobs(){
    jobCollectorSplashAnimationFinished.cancelIt()
    jobCollectorIsConnected.cancelIt()
}

private fun attachObservers(){
    jobCollectorSplashAnimationFinished.renew()
    jobCollectorIsConnected.renew()
}

private fun generateSplashAnimationFinishedCollectorJob() =
lifecycleScope.launchWhenResumed {
    splashAnimationFinished.filter { it == true }.collect {
        withContext(Dispatchers.Main){
            navController?.let {
                if (it.currentDestination?.id == R.id.splashFragment){
                    if (vm.connectivityStatus.value.second is
ConnectivityStatus.Connected.Authenticated){

it.navigate(R.id.action_splashFragment_to_chatListFragment_NO_FILTER)
                    } else {

it.navigate(R.id.action_splashFragment_to_authFragment)
                    }
                }
            }
        }
    }
}

```

```

    }
}

private fun generateIsConnectedCollectorJob() =
lifecycleScope.launchWhenStarted {
    vm.connectivityStatus
        .combine(splashAnimationFinished){ statuses, isFinished -> statuses
to isFinished }
        .dropWhile { val (_, isFinished) = it; !isFinished }
        .collect {
            val (status, _) = it
            val (statusOld, statusNew) = status
            withContext(Dispatchers.Main) {
                if (statusOld::class.java != statusNew::class.java){
                    if (statusOld is
ConnectivityStatus.Connected.Authenticated && statusNew is
ConnectivityStatus.NotConnected){
                        vm.stopClientWorker()
                    }
                    if (statusNew is
ConnectivityStatus.Connected.Authenticated){

navController?.navigate(R.id.chatListFragment_NO_FILTER, null,
navOptionsDefaultSingleTopForMenuItems)
                    } else if (statusNew is
ConnectivityStatus.NotConnected){
                        navController?.navigate(R.id.authFragment, null,
navOptionsDefaultSingleTopForMenuItems)
                    }
                    } else {
                        if (statusNew is
ConnectivityStatus.Connected.Authenticated &&
navController?.currentBackStackEntry?.destination?.id == R.id.authFragment){

navController?.navigate(R.id.chatListFragment_NO_FILTER, null,
navOptionsDefaultSingleTopForMenuItems)
                        }
                    }
                    updateBottomMenuAccordingToIsConnectedState(statusNew)
                }
            }
        }
}
}

```

```

        private fun updateBottomMenuAccordingToIsConnectedState(status:
ConnectivityStatus) {
            if (navController?.currentBackStackEntry?.destination?.id ==
R.id.chatFragment){
                return
            }
            when {
                status is ConnectivityStatus.Connected.Authenticated -> {
                    if (!binding.bottomMenu.isVisible ||
binding.bottomMenu.menu.size != ROUTES_MENU_WITH_CHATS.size) {
                        binding.bottomMenu.menu.clear()
                        binding.bottomMenu.inflateMenu(R.menu.menu_with_chats)
                        hideShowBottomMenu(isVisible = true)
                    }
                }
                status isOneOf listOf(
                    ConnectivityStatus.ChosenServerToConnect::class.java,
                    ConnectivityStatus.Connected.ConnectedToServer::class.java,
                    NotConnected::class.java
                ) -> {
                    if (!binding.bottomMenu.isVisible ||
binding.bottomMenu.menu.size != ROUTES_GENERAL_MENU.size) {
                        binding.bottomMenu.menu.clear()
                        binding.bottomMenu.inflateMenu(R.menu.general_menu)
                        hideShowBottomMenu(isVisible = true)
                    }
                }
            }
        }
    }
}

```

```

package com.edu.alterjuicechat.ui

```

```

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.lifecycle.LifecycleScope
import androidx.recyclerview.widget.LinearLayoutManager
import com.edu.alterjuicechat.databinding.FragmentChatBinding
import com.edu.alterjuicechat.domain.NavArgKeys

```

```

import com.edu.alterjuicechat.domain.model.DataUser
import com.edu.alterjuicechat.extensions.makeGone
import com.edu.alterjuicechat.extensions.scaleUpDown
import com.edu.alterjuicechat.ui.adapters.MessagesAdapter
import com.edu.alterjuicechat.ui.base.BaseFragment
import com.edu.alterjuicechat.viewmodels.ChatViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.collect
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import org.koin.androidx.viewmodel.ext.android.viewModel
import org.koin.core.component.KoinComponent

class ChatFragment :
BaseFragment<FragmentChatBinding>(FragmentChatBinding::inflate), KoinComponent{

    private val userName by lazy {

navController?.currentBackStackEntry?.arguments?.getString(NavArgKeys.CHAT_USER_
NAME)?:: throw IllegalArgumentException("No argument passed for UserName")
    }

    private val userID by lazy{

navController?.currentBackStackEntry?.arguments?.getString(NavArgKeys.CHAT_USER_
ID)?:: throw IllegalArgumentException("No argument passed for userID")
    }

    private val vm by viewModel<ChatViewModel>()

    private var messagesAdapter: MessagesAdapter? = null //by lazy{
MessagesAdapter(userID) }
    // private val chatTitle by lazy{ userName?: Consts.BLANK_TITLE_PLACEHOLDER
}

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }

```

```

    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        (activity as? MainActivity)?.manipulateOnBottomMenu {
            this.makeGone()
        }
        binding.textInputMessage.text.clear()
        binding.buttonSendMessage.setOnClickListener {
            performSendMessage()
        }
    }
}

private var scrolledDownAtStart: Boolean = false

override fun onStart() {
    super.onStart()

    vm.markAsActiveChatUID(DataUser(userID, userName))
    setChatTitle(userName)
    binding.messagesList.apply {
        adapter = MessagesAdapter(userID).also { messagesAdapter = it }
        if (layoutManager == null)
            layoutManager = LinearLayoutManager(context,
LinearLayoutManager.VERTICAL, true)

    }
    messagesAdapter?.itemCount?.takeIf { it != 0 }?.let {
        binding.messagesList.scrollToPosition(it - 1)
        scrolledDownAtStart = true
    }

    lifecycleScope.launch {
        vm.activeChatMessages.collect {
            withContext(Dispatchers.Main){
                messagesAdapter?.submitList(it){
                    if (!scrolledDownAtStart){
                        messagesAdapter?.itemCount?.let {
                            binding.messagesList.scrollToPosition(it - 1)
                            scrolledDownAtStart = true
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}

override fun onDestroy() {
    // Setting unread counter to 0 when exiting chat fragment
    vm.clearUnreadCounter(DataUser(userID, userName))
    super.onDestroy()
}

private fun performSendMessage(){
    val msgText = binding.textInputMessage.text.toString()
    if (msgText.isNotBlank()) {
        vm.sendMessage(DataUser(userID, userName), msgText)
        binding.textInputMessage.text.clear()
        binding.buttonSendMessage.scaleUpDown()
    }
}

private fun setChatTitle(title: String){
    binding.chatHeaderTitle.text = title
}

}

package com.edu.alterjuicechat.ui

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.widget.addTextChangedListener
import androidx.fragment.app.Fragment
import androidx.lifecycle.LifecycleScope
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.utils.generateConfirmationAlertDialog
import com.edu.alterjuicechat.databinding.FragmentUserSettingsBinding
import com.edu.alterjuicechat.domain.UserDisplayName
import com.edu.alterjuicechat.domain.storage.user_data.UserData
import com.edu.alterjuicechat.extensions.animateBackgroundGradient

```

```

import com.edu.alterjuicechat.extensions.viewBinding
import com.edu.alterjuicechat.ui.base.BaseFragment
import com.edu.alterjuicechat.utils.ColorGenerator
import com.edu.alterjuicechat.viewmodels.UserSettingsData
import com.edu.alterjuicechat.viewmodels.UserSettingsViewModel
import kotlinx.coroutines.flow.collect
import kotlinx.coroutines.launch
import org.koin.android.ext.android.get
import org.koin.androidx.viewmodel.ext.android.viewModel

class ProfileSettingsFragment :
BaseFragment<FragmentUserSettingsBinding>(FragmentUserSettingsBinding::inflate)
{
    private val vm by viewModel<UserSettingsViewModel>()
    private val userData = get<UserData>()

    private var dataWithChanges = UserSettingsData(
        userDisplayName = userData.getDisplayName()
    )

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }
    private fun updateProfileColor(input: String) {
        binding.settingsUserColor.background.setTint(
            ColorGenerator.generateIntColorFromInput(input)
        )
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.root.animateBackgroundGradient()
        attachObservers()
        updateProfileColor(dataWithChanges.userDisplayName)
        binding.settingsUserNameInput.addTextChangedListener {
            dataWithChanges = dataWithChanges.copy(userDisplayName =
it?.toString().orEmpty())
            updateProfileColor(dataWithChanges.userDisplayName)

```

```

    }
    binding.generalSettingsButton.setOnClickListener {

navController?.navigate(R.id.action_profileSettingsFragment_to_settingsFragment)
    }
    updateUserSettingsInfoView(userData.getDisplayName())
    binding.userSettingsSaveChanges.setOnClickListener {
        dataWithChanges = dataWithChanges.copy(
            userDisplayName = binding.settingsUserNameInput.text.toString()
        )
        vm.confirmChanges(dataWithChanges)
    }
    binding.userSettingsEditOrCancelButton.setOnClickListener {
        if (vm.isInEditModeState.value){
            dataWithChanges = dataWithChanges.copy(
                userDisplayName =
binding.settingsUserNameInput.text.toString()
            )
            if (vm.checkIfHasChanges(dataWithChanges)){
                // hasChanges
                generateConfirmationAlertDialog(
                    onDiscardChanges = {
                        dataWithChanges =
vm.discardChangesAndCloseEditMode()
                        updateUserSettingsInfoView(
                            userDisplayName =
dataWithChanges.userDisplayName
                        )
                    }
                ).show()
            } else {
                dataWithChanges = vm.discardChangesAndCloseEditMode()
                updateUserSettingsInfoView(
                    userDisplayName = dataWithChanges.userDisplayName
                )
            }
        } else {
            dataWithChanges = vm.enterEditMode()
            updateUserSettingsInfoView(
                userDisplayName = dataWithChanges.userDisplayName
            )
        }
    }
}
}

```

```

    }

    private fun updateUserSettingsInfoView(
        userDisplayName: UserDisplayName? = null
    ) {
        if (userDisplayName != null){
            binding.settingsUserNameInput.setText(userDisplayName)
        }
    }

    private fun attachObservers(){
        lifecycleScope.launch {
            vm.isInEditModeState.collect { isInEditMode ->
                val multiplier = if (isInEditMode) 1 else 0
                binding.userSettingsSaveChanges.isEnabled = isInEditMode
                binding.settingsUserNameInput.isEnabled = isInEditMode
                binding.settingsUserNameLayout.elevation = 10f * multiplier
                if (isInEditMode){
                    binding.userSettingsEditOrCancelButton.text =
getString(R.string.text_cancel)
                } else {
                    binding.userSettingsEditOrCancelButton.text =
getString(R.string.text_edit)
                }
            }
        }
    }
}

package com.edu.alterjuicechat.ui

import com.edu.alterjuicechat.R
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import com.edu.alterjuicechat.BuildConfig
import com.edu.alterjuicechat.databinding.FragmentSettingsBinding

```

```

import com.edu.alterjuicechat.domain.Consts
import com.edu.alterjuicechat.storage.HostDataStorage
import com.edu.alterjuicechat.storage.SessionDynamicDataStorage
import com.edu.alterjuicechat.storage.SessionStaticDataStorage
import com.edu.alterjuicechat.storage.UserDataStorage
import com.edu.alterjuicechat.ui.adapters.SettingsDestinationsAdapter
import com.edu.alterjuicechat.ui.base.BaseFragment
import com.edu.alterjuicechat.ui.model.Destination
import com.edu.alterjuicechat.utils.generateAlertDialog
import java.io.File

const val SHARED_PREFS_DIR = "shared_prefs"

class SettingsFragment :
BaseFragment<FragmentSettingsBinding>(FragmentSettingsBinding::inflate){

    private val settingsDestinationsAdapter by lazy {
        SettingsDestinationsAdapter(
            listOf<Destination>(
                clearDataDestinationCallback,
                openDeveloperInfo
            )
        )
    }

    private val clearDataDestinationCallback by lazy {
        Destination.Callback(getString(R.string.clear_all_data)) {
            generateAlertDialog(
                context,
                getString(R.string.all_data_clear_confirm_title),
                getString(R.string.all_data_clear_description),
                positiveButtonText = getString(R.string.text_confirm),
                negativeButtonText =
getString(R.string.all_data_cancel_clearing),
                onPositiveButtonClick = {
                    invalidateAllData()
                },
            ).show()
        }
    }

    private val openDeveloperInfo by lazy {
        Destination.Callback(getString(R.string.about_app_destination_title)) {

```

```

generateAlertDialog(
    context,
    getString(R.string.about_app_page_title),
    getString(
        R.string.about_app_page_main_content,
        getString(R.string.app_name),
        Consts.APP_GITHUB_CREATOR_SHORT_INFO,
        BuildConfig.VERSION_NAME
    ),
    positiveButtonText = getString(R.string.text_ok),
    negativeButtonText = getString(R.string.open_github),
    onNegativeButtonClick = {
        startActivity(
            Intent(
                Intent.ACTION_VIEW,
                Uri.parse(Consts.APP_GITHUB_CREATOR_LINK)
            )
        )
    }
).show()
}

private fun invalidateAllData(){
    context?.let { context ->
        HostDataStorage.get(context).invalidateAllData()
        SessionStaticDataStorage.get(context).invalidateAllData()
        UserDataStorage.get(context).invalidateAllData()
        val prefsDir = File(context.dataDir, SHARED_PREFS_DIR)
        val allPrefs = if (prefsDir.exists() && prefsDir.isDirectory)
prefsDir.list() else emptyArray()
        val allSessionDynamicPrefs = allPrefs.filter {
            it.startsWith(SessionDynamicDataStorage.fileNamePrefix)
        }.toTypedArray()
        allSessionDynamicPrefs.forEach {
            context.deleteSharedPreferences(it)
        }
        Toast.makeText(context, R.string.all_data_cleared_successfully,
Toast.LENGTH_SHORT).show()
    }
}

override fun onCreateView(

```

```

        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        binding.settingsDestinationsRecyclerView.adapter =
settingsDestinationsAdapter
        binding.backButton.setOnClickListener {
            navController?.navigateUp()
        }
    }
}

package com.edu.alterjuicechat.ui.adapters

import android.graphics.Color
import android.view.View
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.edu.alterjuicechat.databinding.ChatsListItemBinding
import com.edu.alterjuicechat.domain.Consts
import com.edu.alterjuicechat.domain.model.Chat
import com.edu.alterjuicechat.domain.model.User
import com.edu.alterjuicechat.extensions.genBinding
import java.text.SimpleDateFormat
import java.util.*

fun getFormattedDate(milliseconds: Long): String{
    return
SimpleDateFormat(Consts.SIMPLE_MESSAGE_DATE_FORMAT).format(Date(milliseconds))
}

class ChatAdapter(private val onChatClick: (Chat) -> Unit) :
    ListAdapter<Chat, ChatAdapter.ChatHolder>(ChatDifferenceCallback()) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ChatHolder {
        val binding = genBinding(ChatsListItemBinding::inflate, parent)

```

```

        return ChatHolder(binding)
    }

    override fun onBindViewHolder(holder: ChatHolder, position: Int) {
        holder.bind(onChatClick, getItem(position))
    }

    class ChatHolder(private val binding: ChatsListItemBinding) :
        RecyclerView.ViewHolder(binding.root) {

        fun bind(onClick: (Chat) -> Unit, chatItem: Chat) {
            binding.root.setOnClickListener { onClick(chatItem) }
            binding.chatTitle.text = chatItem.userName
            binding.chatLastMessageText.text =
chatItem.lastMessage.message?.takeIf { it.isNotBlank() }?: "History is empty"
            // Show last message date. if date.seconds is 0 -> set view as
invisible
            if (chatItem.lastMessage.dateInMs != 0L) {
                binding.chatLastMessageDate.visibility = View.VISIBLE
                binding.chatLastMessageDate.text =
getFormattedDate(chatItem.lastMessage.dateInMs)
            } else {
                binding.chatLastMessageDate.visibility = View.INVISIBLE
            }
            if (chatItem.isOnline) {
                binding.isOnline.background.setTint(Color.GREEN)
            } else {
                binding.isOnline.background.setTint(Color.RED)
            }

            // Show counter of unread messages. if 0 -> set view as invisible
            if (chatItem.unreadMessagesCount == 0) {
                binding.chatCountUnreadMessages.visibility = View.INVISIBLE
            } else {
                binding.chatCountUnreadMessages.visibility = View.VISIBLE
                binding.chatCountUnreadMessages.text =
chatItem.unreadMessagesCount.toString()
            }
        }
    }

    class ChatDifferenceCallback : DiffUtil.ItemCallback<Chat>() {
        override fun areContentsTheSame(oldItem: Chat, newItem: Chat): Boolean {

```

```

        return oldItem.isOnline == newItem.isOnline &&
            oldItem.lastMessage.message == newItem.lastMessage.message
    &&
        oldItem.unreadMessagesCount == newItem.unreadMessagesCount
    &&
        oldItem.lastMessage.dateInMs == newItem.lastMessage.dateInMs
    }

    override fun areItemsTheSame(oldItem: Chat, newItem: Chat): Boolean {
        return oldItem.userID == newItem.userID
    }
}

```

```

package com.edu.alterjuicechat.ui.adapters

import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import androidx.viewbinding.ViewBinding
import com.edu.alterjuicechat.databinding.MessageItemMineBinding
import com.edu.alterjuicechat.databinding.MessageItemNotMineBinding
import com.edu.alterjuicechat.domain.model.Message
import com.edu.alterjuicechat.extensions.genBinding
import java.text.SimpleDateFormat

private const val VIEW_TYPE_MESSAGE_IS_MINE = 0
private const val VIEW_TYPE_MESSAGE_IS_NOT_MINE = 1

class MessagesAdapter(private val userID: String) : ListAdapter<Message,
MessagesAdapter.BaseMessageViewHolder<ViewBinding>>(MessageDifferenceCallback())
{
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
BaseMessageViewHolder<ViewBinding> {

        return if (viewType == VIEW_TYPE_MESSAGE_IS_MINE){
            MineMessagesHolder(genBinding(MessageItemMineBinding::inflate,
parent))
        } else {
            NotMineMessagesHolder(genBinding(MessageItemNotMineBinding::inflate,
parent))
        } as BaseMessageViewHolder<ViewBinding>
    }
}

```

```

    }

    override fun getItemViewType(position: Int): Int {
        if (getItem(position).senderId == userID)
            return VIEW_TYPE_MESSAGE_IS_NOT_MINE
        return VIEW_TYPE_MESSAGE_IS_MINE
    }

    override fun onBindViewHolder(holder: BaseMessageViewHolder<ViewBinding>,
position: Int) {
        holder.bind(getItem(position))
    }

    class MessageDifferenceCallback : DiffUtil.ItemCallback<Message>() {
        override fun areContentsTheSame(oldItem: Message, newItem: Message):
Boolean {
            return oldItem.senderId == newItem.senderId && oldItem.text ==
newItem.text
        }

        override fun areItemsTheSame(oldItem: Message, newItem: Message):
Boolean {
            return oldItem.senderId == newItem.senderId && oldItem.text ==
newItem.text
        }
    }

    abstract class BaseMessageViewHolder<VB: ViewBinding>(protected val binding:
VB) : RecyclerView.ViewHolder(binding.root){
        companion object{ val sdf = SimpleDateFormat("HH:mm") }
        abstract fun bind(message: Message)
    }

    class MineMessagesHolder(binding: MessageItemMineBinding) :
BaseMessageViewHolder<MessageItemMineBinding>(binding) {
        override fun bind(message: Message) {
            binding.messageText.text = message.text
            binding.messageDate.text = sdf.format(message.date)
        }
    }

```

```

        class NotMineMessagesHolder(binding: MessageItemNotMineBinding) :
            BaseMessageViewHolder<MessageItemNotMineBinding>(binding) {
                override fun bind(message: Message) {
                    binding.messageText.text = message.text
                    binding.messageDate.text = sdf.format(message.date)
                }
            }
    }

package com.edu.alterjuicechat.ui.adapters

import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.edu.alterjuicechat.databinding.SettingsDestinationItemBinding
import com.edu.alterjuicechat.extensions.genBinding
import com.edu.alterjuicechat.ui.model.Destination

class SettingsDestinationsAdapter (
    private val destinations: List<Destination>
): RecyclerView.Adapter<SettingsDestinationViewHolder>() {
    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int
    ): SettingsDestinationViewHolder {
        val binding = genBinding(SettingsDestinationItemBinding::inflate,
parent, false)
        return SettingsDestinationViewHolder(binding)
    }

    override fun onBindViewHolder(holder: SettingsDestinationViewHolder,
position: Int) {
        val destination = destinations.get(position)
        holder.bind(destination)
    }

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

class SettingsDestinationViewHolder(

```

```

        private val binding: SettingsDestinationItemBinding
    ): RecyclerView.ViewHolder(binding.root) {
        fun bind(destination: Destination){
            binding.destinationTitle.text = destination.title
            binding.root.setOnClickListener {
                if (destination is Destination.Callback){
                    destination.onClick.invoke()
                } else if (destination is Destination.ContextCallback){
                    destination.onClick.invoke(it.context)
                }
            }
        }
    }
}

```

```

package com.edu.alterjuicechat.ui.adapters

```

```

import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.edu.alterjuicechat.databinding.AvailableServerItemBinding
import com.edu.alterjuicechat.extensions.genBinding
import com.edu.alterjuicechat.server_model.dto.ServerParams
import com.edu.alterjuicechat.server_model.dto.ServerParamsWithSize

```

```

class AvailableServersAdapter(
    private val onServerClick: (ServerParams) -> Unit
): ListAdapter<ServerParamsWithSize,
    AvailableServersViewHolder>(ServersDifferenceCallback()){

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
    AvailableServersViewHolder {
        return
        AvailableServersViewHolder(genBinding(AvailableServerItemBinding::inflate,
        parent))
    }
}

```

```

        override fun onBindViewHolder(holder: AvailableServersViewHolder, position:
Int) {
            holder.bind(getItem(position), onServerClick)
        }

        class ServersDifferenceCallback :
DiffUtil.ItemCallback<ServerParamsWithSize>() {
            override fun areContentsTheSame(
                oldItem: ServerParamsWithSize,
                newItem: ServerParamsWithSize
            ): Boolean {
                return oldItem.ip == newItem.ip && oldItem.clientsSize ==
newItem.clientsSize && oldItem.name == newItem.name
            }

            override fun areItemsTheSame(
                oldItem: ServerParamsWithSize,
                newItem: ServerParamsWithSize
            ): Boolean {
                return oldItem.ip == newItem.ip && oldItem.name == newItem.name
            }
        }
    }

    class AvailableServersViewHolder(
        private val binding: AvailableServerItemBinding
    ): RecyclerView.ViewHolder(binding.root){

        fun bind(serverInfo: ServerParamsWithSize, onServerClick:
(ServerParamsWithSize) -> Unit) {
            binding.serverName.text = serverInfo.name
            binding.serverIP.text = serverInfo.ip
            serverInfo.clientsSize.toString().let {
                binding.clientsCount.text = it
                binding.clientsCountCircled.text = it
            }
            binding.root.setOnClickListener {
                onServerClick(serverInfo)
            }
        }
    }
}

```

```

package com.edu.alterjuicechat.ui

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.lifecycle.LifecycleScope
import com.edu.alterjuicechat.R
import
com.edu.alterjuicechat.utils.generateConnectionConfirmationAlertDialogToChosenSe
rverInfo
import com.edu.alterjuicechat.databinding.FragmentFoundServersListBinding
import com.edu.alterjuicechat.extensions.startMyForegroundService
import com.edu.alterjuicechat.extensions.stopMyForegroundService
import com.edu.alterjuicechat.server_model.dto.ServerParams
import com.edu.alterjuicechat.services.ConnectedToServerService
import com.edu.alterjuicechat.services.ServersFinderService
import com.edu.alterjuicechat.ui.adapters.AvailableServersAdapter
import com.edu.alterjuicechat.ui.base.BaseFragment
import com.edu.alterjuicechat.viewmodels.FoundServerListViewModel
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.*
import org.koin.androidx.viewmodel.ext.android.viewModel

class FoundServersListFragment :
BaseFragment<FragmentFoundServersListBinding>(FragmentFoundServersListBinding::i
nflate) {
    private val vm by viewModel<FoundServerListViewModel>()

    private val availableServersAdapter by lazy {
        AvailableServersAdapter(::onServerClick)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

```

```

        super.onViewCreated(view, savedInstanceState)
        attachObservers()
        binding.availableServersRecycler.adapter = availableServersAdapter
        binding.imageUpdateChats.setOnClickListener {
            Toast.makeText(context, R.string.toast_users_updated,
Toast.LENGTH_SHORT).show()
        }
    }

    private fun onServerClick(params: ServerParams){
        generateConnectionConfirmationAlertDialogToChosenServerInfo(
            serverName = params.name,
            onPositiveButtonClick = {
                stopMyForegroundService<ServersFinderService>(requireContext())
                vm.connectToServer(params)
            }
        )
        startMyForegroundService<ConnectedToServerService>(requireContext())
    }

    private fun attachObservers(){
        lifecycleScope.launch(Dispatchers.IO) {
            vm.availableServers.collectLatest { found ->
                withContext(Dispatchers.Main){
                    availableServersAdapter.submitList(found)
                }
            }
        }
    }

}

package com.edu.alterjuicechat.ui.model

enum class ChatsFilter {
    NO_FILTER, FILTER_ACTIVE
}

package com.edu.alterjuicechat.ui.model

import android.content.Context

```

```
sealed class Destination(val title: String) {
    class Callback(title: String, val onClick: () -> Unit): Destination(title)
    class ContextCallback(title: String, val onClick: (Context) -> Unit):
Destination(title)
}
```

```
package com.edu.alterjuicechat.ui.custom.wifi_waves
```

```
import android.animation.ValueAnimator
import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF
import android.util.AttributeSet
import android.view.View
import android.view.animation.LinearInterpolator
import com.alterjuice.utils.half
import com.alterjuice.utils.whiledIndexed
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.extensions.makeInvisible
import com.edu.alterjuicechat.extensions.makeVisible
import kotlin.math.hypot
import kotlin.math.max
```

```
private const val MAX_ALPHA = 255
```

```
private const val CALCULATE_BY_ALLOWED_SPACE = 0
```

```
private const val CALCULATE_BY_WAVES_AMOUNT = 1
```

```
class WiFiWavesView: View, WiFiWavesController {
    private var waveIncreasingDuration: Long = 900L

    // Attrs
    private var waveColor = context.getColor(R.color.defaultTextColor)
    private var waveSpaceStep: Float = 40f
    private var initialRadius = 20f
    private var wavesAmount = 10
    private var waveWidth = 2f
    private var makeWavesFade = false
    private var fadeOffset = 3
    private var wavesAreVisible = true
```

```

private var drawBy = CALCULATE_BY_WAVES_AMOUNT

// Paints
private var wavePaint: Paint = getWavePaint()

// Other variables
private var center = PointF(0f, 0f)
private var circlesDrawer: WavesDrawer? = null
private var waveAnimator: ValueAnimator? = null
private var waveMaxRadius = 0f
private var waveRadiusOffset = 0f
    set(value) {
        field = value
        postInvalidateOnAnimation()
    }

private fun getWavePaint(): Paint {
    return Paint(Paint.ANTI_ALIAS_FLAG).apply {
        color = waveColor
        strokeWidth = waveWidth
        style = Paint.Style.STROKE
        alpha = MAX_ALPHA
    }
}

private fun invalidatePaints(){
    wavePaint = getWavePaint()
}

constructor(context: Context) : super(context) {
    init(null, 0)
}

constructor(context: Context, attrs: AttributeSet) : super(context, attrs) {
    init(attrs, 0)
}

constructor(context: Context, attrs: AttributeSet, defStyle: Int) :
super(context, attrs, defStyle) {
    init(attrs, defStyle)
}

```

```

private fun getDrawerAccordingToDrawByOption(calculateBy: Int): WavesDrawer
{
    return when (calculateBy) {
        CALCULATE_BY_WAVES_AMOUNT -> drawerByCountCircles
        CALCULATE_BY_ALLOWED_SPACE -> drawerByAllowedSpaceDrawer
        else -> drawerByAllowedSpaceDrawer
    }
}

private fun init(attrs: AttributeSet?, defStyle: Int) {
    val attrs = context.obtainStyledAttributes(attrs,
R.styleable.WiFiWavesView, defStyle, 0)

    waveColor = attrs.getInt(R.styleable.WiFiWavesView_waveColor, waveColor)
    waveSpaceStep =
attrs.getDimension(R.styleable.WiFiWavesView_waveSpaceStep, waveSpaceStep)
    initialRadius =
attrs.getDimension(R.styleable.WiFiWavesView_initialRadius, initialRadius)
    wavesAmount = attrs.getInt(R.styleable.WiFiWavesView_wavesAmount,
wavesAmount)
    waveWidth = attrs.getFloat(R.styleable.WiFiWavesView_wavewidth,
waveWidth)
    makeWavesFade =
attrs.getBoolean(R.styleable.WiFiWavesView_makeWavesFade, makeWavesFade)
    fadeOffset = attrs.getInt(R.styleable.WiFiWavesView_fadeOffset,
fadeOffset)
    drawBy = attrs.getInt(R.styleable.WiFiWavesView_drawBy, drawBy)

    wavesAreVisible =
attrs.getBoolean(R.styleable.WiFiWavesView_wavesAreVisible, wavesAreVisible)
    if (wavesAreVisible){
        showWaves()
    } else {
        hideWaves()
    }

    invalidatePaints()
    attrs.recycle()
}

override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
    center.set(w.half() , h.half())
}

```

```

        waveMaxRadius = hypot(center.x.toDouble(),
center.y.toDouble()).toFloat()
    }

    override fun onDetachedFromWindow() {
        waveAnimator?.cancel()
        super.onDetachedFromWindow()
    }

    override fun onAttachedToWindow() {
        super.onAttachedToWindow()
        setAndStartWaveAnimator()
    }

    private fun clearWaveAnimator(){
        waveAnimator?.cancel()
        waveAnimator = null
    }

    private fun setAndStartWaveAnimator(){
        waveAnimator = generateWaveAnimator().also {
            it.start()
        }
    }

    private fun generateWaveAnimator() = ValueAnimator.ofFloat(0f,
waveSpaceStep).also {
        it.duration = waveIncreasingDuration
        it.repeatMode = ValueAnimator.RESTART
        it.repeatCount = ValueAnimator.INFINITE
        it.interpolator = LinearInterpolator()
        it.addUpdateListener { animator ->
            waveRadiusOffset = animator.animatedValue as Float
        }
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        circlesDrawer?.draw(canvas)
    }

    /** An implementation of WavesDrawer.
    * It helps to draw the waves according the allowed space.

```

```

* */
private val drawerByAllowedSpaceDrawer = object : WavesDrawer.ByAllowedSpace
{
    override fun draw(canvas: Canvas) {
        var currentRadius = initialRadius + waveRadiusOffset
        val wavesWillBeDrawn = ((waveMaxRadius - currentRadius) /
waveSpaceStep).toInt()

        val alphaStep = if (makeWavesFade){
            (MAX_ALPHA / wavesWillBeDrawn)
        } else { 0 }

        val alphaWavesOffset = if (makeWavesFade){
            fadeOffset
        } else { 0 }

        whileIndexed { currentRadius < waveMaxRadius }() { index ->
            canvas.drawCircle(center.x, center.y, currentRadius, wavePaint)
            currentRadius += waveSpaceStep
            wavePaint.alpha = calculateNextAlpha(index, alphaStep,
alphaWavesOffset)
        }
        wavePaint.alpha = MAX_ALPHA
    }
}

/** An implementation of WavesDrawer.
 * It helps to draw the waves according the amount of waves
 * */
private val drawerByCountCircles = object: WavesDrawer.ByCountCircles {
    override fun draw(canvas: Canvas) {
        val alphaStep = if (makeWavesFade){
            (MAX_ALPHA / wavesAmount)
        } else { 0 }

        val alphaWavesOffset = if (makeWavesFade){
            fadeOffset
        } else { 0 }

        var currentRadius = initialRadius + waveRadiusOffset

        for (index in 0..wavesAmount) {
            canvas.drawCircle(center.x, center.y, currentRadius, wavePaint)

```

```

        currentRadius += waveSpaceStep
        wavePaint.alpha = calculateNextAlpha(index, alphaStep,
alphaWavesOffset)
    }
    wavePaint.alpha = MAX_ALPHA
}

}

/** Calculates the alpha for the wave according to index */
private fun calculateNextAlpha(index: Int, alphaStep: Int, alphaWavesOffset:
Int): Int {
    return max(0, MAX_ALPHA - (alphaStep * (index + alphaWavesOffset)))
}

override fun showWaves() {
    if (isAlreadyVisible()) return
    this.circlesDrawer = getDrawerAccordingToDrawByOption(drawBy)
    setAndStartWaveAnimator()
    (this as View).makeVisible()
}

override fun hideWaves() {
    if (!isAlreadyVisible()) return
    this.circlesDrawer = null
    clearWaveAnimator()
    (this as View).makeInvisible()
}

override fun isAlreadyVisible(): Boolean {
    return circlesDrawer != null
}

override fun showHideWaves(isVisible: Boolean) {
    if (isAlreadyVisible() == isVisible) return
    when (isVisible) {
        true -> showWaves()
        false -> hideWaves()
    }
}
}
}

```

```
package com.edu.alterjuicechat.ui.custom.wifi_waves
```

```

interface WiFiWavesController {
    fun showWaves()
    fun hideWaves()
    fun isVisible(): Boolean
    fun showHideWaves(isVisible: Boolean)
}

package com.edu.alterjuicechat.ui.custom.wifi_waves

import android.graphics.Canvas

internal sealed interface WavesDrawer {
    fun draw(canvas: Canvas)
    interface ByAllowedSpace : WavesDrawer
    interface ByCountCircles : WavesDrawer
}

package com.edu.alterjuicechat.ui.base

import android.view.LayoutInflater
import androidx.viewbinding.ViewBinding

abstract class LaunchFragment<T: ViewBinding>(
    bindingInflater: (LayoutInflater) -> T
): BaseFragment<T>(bindingInflater) {

    fun notifySplashAnimationFinished(){
        if (isAdded) {
            with(requireActivity() as? LaunchActivity) {
                this?.notifySplashAnimationFinished()
            }
        }
    }
}

package com.edu.alterjuicechat.ui.base

import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.NavController
import androidx.navigation.NavHost
abstract class BaseActivity : AppCompatActivity() {

```

```

        var navHost: NavHost? = null
        val navController: NavController? get() = navHost?.navController
    }

```

```
package com.edu.alterjuicechat.ui.base
```

```

import android.view.LayoutInflater
import androidx.fragment.app.Fragment
import androidx.viewbinding.ViewBinding
import com.edu.alterjuicechat.extensions.viewBinding

```

```

abstract class BaseFragment<T: ViewBinding>(
    bindingInflater: (LayoutInflater) -> T
): Fragment() {
    protected val binding by viewBinding(bindingInflater)

    val navHost get() = (activity as? BaseActivity)?.navHost
    val navController get() = navHost?.navController
}

```

```
package com.edu.alterjuicechat.ui.base
```

```

import androidx.lifecycle.LifecycleScope
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.launch

```

```

abstract class LaunchActivity : BaseActivity() {
    protected val splashAnimationFinished = MutableSharedFlow<Boolean>()
    fun notifySplashAnimationFinished(){
        lifecycleScope.launch {
            splashAnimationFinished.emit(true)
        }
    }
}

```

```
package com.edu.alterjuicechat.viewmodels
```

```

import androidx.lifecycle.ViewModel
import com.edu.alterjuicechat.domain.UserDisplayName
import com.edu.alterjuicechat.domain.storage.user_data.UserData

```

```

import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow

abstract class UserSettingsViewModel: ViewModel(){
    abstract val isInEditModeState: StateFlow<Boolean>

    abstract fun checkIfHasChanges(dataWithChanges: UserSettingsData): Boolean
    abstract fun confirmChanges(dataWithChanges: UserSettingsData)

    abstract fun enterEditMode(): UserSettingsData
    abstract fun discardChangesAndCloseEditMode(): UserSettingsData
}

data class UserSettingsData(
    val userDisplayName: UserDisplayName,
)

internal class UserSettingsViewModelImpl(
    private val userData: UserData,
): UserSettingsViewModel() {
    override val isInEditModeState = MutableStateFlow<Boolean>(false)

    private var userSettingsData = generateActualUserSettingsData()

    private fun generateActualUserSettingsData(): UserSettingsData {
        return UserSettingsData(
            userDisplayName = userData.getDisplayName()
        )
    }

    private fun getTransformedActualUserSettingsData(): UserSettingsData {
        return generateActualUserSettingsData()
    }

    override fun checkIfHasChanges(dataWithChanges: UserSettingsData): Boolean {
        if (dataWithChanges.userDisplayName !=
this.userSettingsData.userDisplayName)
            return true
        return false
    }
}

```

```

        override fun confirmChanges(dataWithChanges: UserSettingsData) {
            isInEditModeState.value = false

            userSettingsData = userSettingsData.copy(
                userDisplayName = dataWithChanges.userDisplayName,
            )

            userData.setDisplayName(userSettingsData.userDisplayName)
        }

        override fun enterEditMode(): UserSettingsData {
            isInEditModeState.value = true
            return getTransformedActualUserSettingsData()
        }

        override fun discardChangesAndCloseEditMode(): UserSettingsData {
            isInEditModeState.value = false
            val transformed = userSettingsData.copy()
            return transformed
        }
    }

}

package com.edu.alterjuicechat.viewmodels

import androidx.lifecycle.ViewModel
import com.edu.alterjuicechat.domain.storage.user_data.UserData
import com.edu.alterjuicechat.socket.ServersFinderState
import kotlinx.coroutines.flow.Flow

abstract class AuthViewModel: ViewModel(), ServersFinderState {
    abstract fun getSavedUsername(): String
}

class AuthViewModelImpl(
    private val userData: UserData,
    finder: ServersFinderState,
) : AuthViewModel() {

```

```

        override val availableServers = finder.availableServers
        override val serverScannerIsWorking = finder.serverScannerIsWorking

        override fun getSavedUsername(): String {
            return userData.getDisplayName()
        }
    }
}

package com.edu.alterjuicechat.viewmodels

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.alterjuice.utils.RenewableJob
import com.edu.alterjuicechat.domain.model.ServerInfo
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import com.edu.alterjuicechat.repo.ClientWorker
import com.edu.alterjuicechat.repo.RuntimeConnectivity
import kotlinx.coroutines.delay
import kotlinx.coroutines.flow.*
import kotlinx.coroutines.launch

abstract class MainActivityViewModel: ViewModel() {
    abstract val connectivityStatus: StateFlow<Pair<ConnectivityStatus,
ConnectivityStatus>>

    abstract fun stopClientWorker()
    abstract fun stopAllJobs()
}

class MainActivityViewModelImpl(
    private val clientWorker: ClientWorker,
): MainActivityViewModel() {
    override val connectivityStatus =
MutableStateFlow(Pair(RuntimeConnectivity.statusState.value,
RuntimeConnectivity.statusState.value))
    private var isConnectedCheckerJob =
RenewableJob(::generateIsConnectedCheckerJob)
    private var lastConnectivityCheck: ConnectivityStatus =
connectivityStatus.value.first

    init {

```

```

        isConnectedCheckerJob.renew()
    }

    private fun generateIsConnectedCheckerJob() = viewModelScope.launch {
        while (true) {
            val newStatus = RuntimeConnectivity.statusState.value
            connectivityStatus.emit(Pair(lastConnectivityCheck, newStatus))
            lastConnectivityCheck = newStatus
            delay(2_500)
        }
    }

    override fun stopClientWorker() {
        viewModelScope.launch {
            clientWorker.stopAll()
        }
    }

    override fun stopAllJobs() {
        isConnectedCheckerJob.cancelIt()
    }
}

package com.edu.alterjuicechat.viewmodels

import androidx.lifecycle.ViewModel
import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.model.Chat
import com.edu.alterjuicechat.repo.ClientWorker
import com.edu.alterjuicechat.repo.GlobalFlows
import kotlinx.coroutines.flow.StateFlow

abstract class ChatListViewModel: ViewModel() {
    abstract val allChats: StateFlow<List<Chat>>
    abstract fun loadUsers()
    abstract fun setActiveChatUID(userUID: UserUID?)
}

internal class ChatListViewModelImpl(
    private val clientWorker: ClientWorker,
) : ChatListViewModel() {

    init { loadUsers() }

```

```

        override val allChats: StateFlow<List<Chat>> = GlobalFlows.allChats

        override fun loadUsers() = clientWorker.recreateUsersRequester()
        override fun setActiveChatUID(userUID: UserUID?) =
clientWorker.setActiveChat(userUID)
    }

package com.edu.alterjuicechat.viewmodels

import androidx.lifecycle.ViewModel
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import com.edu.alterjuicechat.repo.RuntimeConnectivity
import com.edu.alterjuicechat.server_model.dto.ServerParams
import com.edu.alterjuicechat.socket.ServersFinderState
import org.koin.core.component.KoinComponent

abstract class FoundServerListViewModel: ViewModel(), ServersFinderState {
    abstract fun connectToServer(params: ServerParams)
}

class FoundServerListViewModelImpl(
    // private val sessionStaticData: SessionStaticData,
    finderState: ServersFinderState,
): FoundServerListViewModel(), KoinComponent {
    override val availableServers = finderState.availableServers
    override val serverScannerIsWorking get() = throw NotImplementedError("Not
used")

    override fun connectToServer(params: ServerParams) {
        val chosenServerToConnect =
ConnectivityStatus.ChosenServerToConnect(params)
        RuntimeConnectivity.emitNewStatus(chosenServerToConnect)
    }
}

package com.edu.alterjuicechat.viewmodels

import androidx.lifecycle.ViewModel
import com.edu.alterjuicechat.domain.model.Message
import com.edu.alterjuicechat.domain.model.User
import com.edu.alterjuicechat.repo.ClientWorker
import com.edu.alterjuicechat.repo.GlobalFlows

```

```

import kotlinx.coroutines.flow.*

abstract class ChatViewModel: ViewModel() {
    abstract val activeChatMessages: StateFlow<List<Message>>
    abstract fun sendMessage(user: User, message: String)
    abstract fun clearUnreadCounter(user: User)
    abstract fun markAsActiveChatUID(user: User)
}

class ChatViewModelImpl(
    private val clientWorker: ClientWorker
) : ChatViewModel() {
    override val activeChatMessages = GlobalFlows.activeChatMessages

    override fun sendMessage(user: User, message: String) =
clientWorker.sendMessage(user, message)
    override fun clearUnreadCounter(user: User) =
clientWorker.clearUnreadCounter(user.userID)

    override fun markAsActiveChatUID(user: User) {
        if (GlobalFlows.activeChatUID.value != user.userID){
            clientWorker.setActiveChat(user.userID)
        }
    }
}

package com.edu.alterjuicechat.viewmodels

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.alterjuice.utils.generateUID
import com.edu.alterjuicechat.domain.ServerName
import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.domain.model.HostStatus
import com.edu.alterjuicechat.domain.storage.server_data.HostData
import com.edu.alterjuicechat.domain.storage.server_data.HostDataStateNonNull
import com.edu.alterjuicechat.repo.RuntimeHost
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

private const val VISIBLE_CHARACTERS_AMOUNT = 9
private const val SERVER_UID_REPLACEMENT = "*****"

```

```

abstract class HostSettingsViewModel: ViewModel(){
    abstract val hostStatus: StateFlow<HostStatus>
    abstract val serverUIDToShow: StateFlow<String>
    abstract val serverNameToShow: StateFlow<ServerName>
    abstract val isInEditModeState: StateFlow<Boolean>

    abstract fun checkIfHasChanges(dataWithChanges: HostDataStateNonNull):
Boolean
    abstract fun confirmChanges(dataWithChanges: HostDataStateNonNull)

    abstract fun enterEditMode(): HostDataStateNonNull
    abstract fun discardChangesAndCloseEditMode(): HostDataStateNonNull

    abstract fun generateNewUID(): ServerUID
    abstract fun updateServerName(newServerName: ServerName)
}

internal class HostSettingsViewModelImpl(
    private val hostData: HostData,
): HostSettingsViewModel() {
    override val hostStatus = RuntimeHost.statusState
    override val serverUIDToShow = MutableStateFlow<String>("")
    override val serverNameToShow = MutableStateFlow<String>("")
    override val isInEditModeState = MutableStateFlow<Boolean>(false)

    private var serverSettingsData = generateActualServerSettingsData()

    init {
        emitAllDataAsActual(serverSettingsData)
    }

    private fun generateActualServerSettingsData(): HostDataStateNonNull {
        return HostDataStateNonNull(
            serverUID = hostData.getServerUID(),
            serverName = hostData.getServerName()
        )
    }

    private fun getTransformedActualServerSettingsData(): HostDataStateNonNull {

```

```

        val data = generateActualServerSettingsData()
        return data.copy(serverUID = transformServerUIDToShow(data.serverUID))
    }

    override fun checkIfHasChanges(dataWithChanges: HostDataStateNonNull):
Boolean {
        if (dataWithChanges.serverName != this.serverSettingsData.serverName)
            return true
        if (dataWithChanges.serverUID != serverSettingsData.serverUID){
            if (!dataWithChanges.serverUID.contains(SERVER_UID_REPLACEMENT)){
                return true
            }
        }
        return false
    }

    override fun confirmChanges(dataWithChanges: HostDataStateNonNull) {
        isInEditModeState.value = false

        serverSettingsData = serverSettingsData.copy(
            serverName = dataWithChanges.serverName,
            serverUID = dataWithChanges.serverUID
        )

        hostData.setServerName(serverSettingsData.serverName)
        hostData.setServerUID(serverSettingsData.serverUID)

        emitAllDataAsActual(serverSettingsData)
    }

    private fun emitAllDataAsActual(serverSettingsData: HostDataStateNonNull){
        emitNewServerUIDToShow(serverSettingsData.serverUID)
        emitNewServerNameToShow(serverSettingsData.serverName)
    }

    override fun enterEditMode(): HostDataStateNonNull {
        isInEditModeState.value = true
        return getTransformedActualServerSettingsData()
    }

    override fun discardChangesAndCloseEditMode(): HostDataStateNonNull {
        isInEditModeState.value = false
    }

```

```

        val transformed = serverSettingsData.copy(
            serverUID = transformServerUIDToShow(serverSettingsData.serverUID)
        )
        emitAllDataAsActual(transformed)
        return transformed
    }

    override fun generateNewUID(): ServerUID {
        return generateUID()
    }

    private fun emitNewServerNameToShow(newServerName: ServerName){
        viewModelScope.launch {
            serverNameToShow.emit(newServerName)
        }
    }

    override fun updateServerName(newServerName: ServerName) {
        if (isServerNameValid(newServerName)){
            hostData.setServerName(newServerName)
            emitNewServerNameToShow(newServerName)
        }
    }

    private fun emitNewServerUIDToShow(newServerUID: ServerUID) {
        viewModelScope.launch {
            serverUIDToShow.emit(transformServerUIDToShow(newServerUID))
        }
    }

    private fun isServerNameValid(serverName: ServerName): Boolean {
        return true
    }

    private fun transformServerUIDToShow(serverUID: ServerUID): String {
        return serverUID.replaceRange(
            range = VISIBLE_CHARACTERS_AMOUNT until serverUID.length-
VISIBLE_CHARACTERS_AMOUNT,
            replacement = SERVER_UID_REPLACEMENT
        )
    }
}

```

```

package com.edu.alterjuicechat

import android.app.Application
import com.edu.alterjuicechat.di.Modules
import
com.edu.alterjuicechat.notifications.notification_creator.checkNotificationChannels
import org.koin.android.ext.koin.androidContext
import org.koin.core.context.startKoin

class App : Application(){

    override fun onCreate() {
        super.onCreate()
        checkNotificationChannels(this)
        startKoin{
            androidContext(this@App)
            modules(Modules.allModules)
        }
    }
}

```

```

package com.edu.alterjuicechat.di

import com.edu.alterjuicechat.domain.storage.server_data.HostData
import
com.edu.alterjuicechat.domain.storage.session_data.dynamic.SessionDynamicData
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import
com.edu.alterjuicechat.domain.storage.session_data.static.SessionStaticData
import com.edu.alterjuicechat.domain.storage.user_data.UserData
import com.edu.alterjuicechat.repo.ClientWorker
import com.edu.alterjuicechat.repo.DataRepo
import com.edu.alterjuicechat.server_model.dto.GeneratorDto
import com.edu.alterjuicechat.server_model.dto.ParserDto
import com.edu.alterjuicechat.socket.*
import com.edu.alterjuicechat.storage.*
import com.edu.alterjuicechat.viewmodels.*
import com.edu.alterjuicechat.viewmodels.HostSettingsViewModelImpl
import org.koin.android.ext.koin.androidContext
import org.koin.androidx.viewmodel.dsl.viewModel

```

```

import org.koin.core.parameter.parametersOf
import org.koin.dsl.module
import com.edu.alterjuicechat.socket.DI as socketDI
import com.edu.alterjuicechat.server_model.DI as serverDI

object Modules {

    private val storageDataModule = module {
        single<HostData> { get<HostDataStorage>() }
        single<SessionStaticData> { get<SessionStaticDataStorage>() }
        single<UserData> { get<UserDataStorage>() }
        single<SessionDynamicData> { (status: ConnectivityStatus.Connected) ->
get<SessionDynamicDataStorage>(){ parametersOf(status)} }
    }

    private val storageModule = module {
        single<HostDataStorage> { HostDataStorage.get(androidContext()) }
        single<SessionStaticDataStorage> {
SessionStaticDataStorage.get(androidContext()) }
        single<UserDataStorage> { UserDataStorage.get(androidContext()) }
        single<SessionDynamicDataStorage> { (status:
ConnectivityStatus.Connected) -> SessionDynamicDataStorage.get(androidContext(),
status) }
    }

    private val activityViewModels = module {
        viewModel<MainActivityViewModel> {
MainActivityViewModelImpl(get<ClientWorker>()) }
    }

    private val providers = module {
        factory<DataRepo> {
            (status: ConnectivityStatus.Connected) ->
DataRepo.get(androidContext(), status, get<UserData>().getDisplayName())
        }

        single<ClientWorker> {
            ClientWorker(
                parser = get<ParserDto>(),
                generator = get<GeneratorDto>(),
                userData = get<UserDataStorage>(),
            ) }
    }
}

```

```

    }

    private val viewModelsModule = module {
        viewModel<ChatListViewModel> {
ChatListViewModelImpl(get<ClientWorker>()) }
        viewModel<ChatViewModel> { ChatViewModelImpl(get<ClientWorker>()) }
        viewModel<AuthViewModel> { AuthViewModelImpl(get<UserData>(),
get<ServersFinderState>()) }
        viewModel<HostSettingsViewModel> {
HostSettingsViewModelImpl(get<HostData>()) }
        viewModel<UserSettingsViewModel> {
UserSettingsViewModelImpl(get<UserData>()) }
        viewModel<FoundServerListViewModel> { FoundServerListViewModelImpl(
            get<ServersFinderState>()
        ) }
    }

    val allModules = listOf(
        providers,
        socketDI.serversFinder,
        storageDataModule,
        storageModule,
        activityViewModels,
        serverDI.modules,
        viewModelsModule
    )
}

package com.edu.alterjuicechat.navigation

import android.app.Activity
import android.app.PendingIntent
import android.content.Context
import android.os.Bundle
import androidx.annotation.IdRes
import androidx.annotation.NavigationRes
import androidx.core.os.bundleOf
import androidx.navigation.NavDeepLinkBuilder
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.domain.NavArgKeys
import com.edu.alterjuicechat.domain.UserDisplayName

```

```

import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.services.ServersFinderService
import com.edu.alterjuicechat.services.getIntentToStopService
import com.edu.alterjuicechat.ui.MainActivity

fun <TActivity: Activity> generatePendingIntentForNavigation(
    context: Context,
    activityClass: Class<TActivity>,
    @NavigationRes navigationGraphID: Int,
    @IdRes destinationResIDs: List<Int>,
    args: Bundle = bundleOf()
): PendingIntent {
    /*
     * How to open fragment page, when pressed a notification in android
     * https://stackoverflow.com/a/55245317 */
    return NavDeepLinkBuilder(context)
        .setComponentName(activityClass)
        .setGraph(navigationGraphID)
        .apply {
            destinationResIDs.forEach(::addDestination)
        }
        .setArguments(args)
        .createPendingIntent()
}

fun generatePendingIntentToViewScannedServersDestination(context: Context):
PendingIntent {
    return generatePendingIntentForNavigation(
        context = context,
        activityClass = MainActivity::class.java,
        navigationGraphID = R.navigation.nav_graph,
        destinationResIDs = listOf(R.id.foundServersListFragment)
    )
}

fun generatePendingIntentToStopScanningServers(context: Context): PendingIntent
{
    val stopSelf = getIntentToStopService<ServersFinderService>(context)
    return PendingIntent.getService(context, 0, stopSelf,
PendingIntent.FLAG_CANCEL_CURRENT or PendingIntent.FLAG_IMMUTABLE)
}

```

```

fun generatePendingIntentForHostSettingsDestination(context: Context):
PendingIntent {
    return generatePendingIntentForNavigation(
        context = context,
        activityClass = MainActivity::class.java,
        navigationGraphID = R.navigation.nav_graph,
        destinationResIDs = listOf(R.id.serverSettingsFragment)
    )
}

fun generatePendingIntentForDirectChatDestination(context: Context, userID:
UserUID, userName: UserDisplayName): PendingIntent {
    return generatePendingIntentForNavigation(
        context = context,
        activityClass = MainActivity::class.java,
        navigationGraphID = R.navigation.nav_graph,
        destinationResIDs = listOf(R.id.nav_graph, R.id.chatFragment),
        args = bundleOf(
            NavArgKeys.CHAT_USER_ID to userID,
            NavArgKeys.CHAT_USER_NAME to userName,
        )
    )
}

package com.edu.alterjuicechat.navigation.data

import com.edu.alterjuicechat.R

val ROUTES_GENERAL_MENU = listOf<Int>(
    R.id.authFragment,
    R.id.profileSettingsFragment,
    R.id.serverSettingsFragment,
)
val ROUTES_MENU_WITH_CHATS = listOf<Int>(
    R.id.chatsList, // chatsList without filters
    R.id.chatsList, // chatsList with just Online users filter
    R.id.profileSettingsFragment,
    R.id.serverSettingsFragment,
)

package com.edu.alterjuicechat.navigation.data

import androidx.navigation.NavOptions

```

```
import com.edu.alterjuicechat.R

val navOptionsWithNoHistory = NavOptions.Builder().setPopUpTo(R.id.nav_graph,
inclusive = true).build()
val navOptionsDefaultSingleTopForMenuItems = NavOptions.Builder()
    .setLaunchSingleTop(true)
    .setEnterAnim(R.anim.fragment_fade_in)
    .setExitAnim(R.anim.fragment_fade_out)
    .setPopEnterAnim(R.anim.fragment_fade_in)
    .setPopExitAnim(R.anim.fragment_fade_out)
    .setPopUpTo(R.id.nav_graph, inclusive = true)
    .build()
```

```
package com.edu.alterjuicechat.utils
```

```
import android.app.ActivityManager
import android.graphics.Color
```

```
object ColorGenerator {
    fun generateStrColorFromInput(input: String?): String {
        if (input == null || input == "") return "#000000"
        val hash = input.hashCode().toString().replace("-", "").toInt()
        val color = String.format("#%06X", hash)
        return color.substring(0, 7)
    }

    fun generateIntColorFromInput(input: String?): Int {
        return Color.parseColor(generateStrColorFromInput(input))
    }

    fun generateColorFromInput(input: String?): Color {
        return Color.valueOf(generateIntColorFromInput(input))
    }
}
```

```
package com.edu.alterjuicechat.utils
```

```
import android.app.AlertDialog
import android.content.Context
import androidx.fragment.app.Fragment
import com.edu.alterjuicechat.R
```

```

import com.edu.alterjuicechat.domain.ServerName

fun generateAlertDialog(
    context: Context?,
    title: String, message: String,
    positiveButtonText: String, onPositiveButtonClick: () -> Unit = {},
    negativeButtonText: String, onNegativeButtonClick: () -> Unit = {},
): AlertDialog = AlertDialog.Builder(context)
    .setTitle(title)
    .setMessage(message)
    .setPositiveButton(positiveButtonText){ dialog, _ ->
        onPositiveButtonClick()
        dialog.cancel()
    }
    .setNegativeButton(negativeButtonText){ dialog, _ ->
        onNegativeButtonClick()
        dialog.dismiss()
    }.create()

fun generateConnectionConfirmationAlertDialogToChosenServerInfo(
    context: Context,
    serverName: ServerName,
    onPositiveButtonClick: () -> Unit = {},
    onNegativeButtonClick: () -> Unit = {},
): AlertDialog = generateAlertDialog(
    context,
    title = context.getString(R.string.confirm_connecting),
    message = context.getString(R.string.are_you_sure_you_want_to_connect_to,
serverName),
    positiveButtonText = context.getString(R.string.text_connect),
onPositiveButtonClick = onPositiveButtonClick,
    negativeButtonText = context.getString(R.string.text_cancel),
onNegativeButtonClick = onNegativeButtonClick
)

fun Fragment.generateConnectionConfirmationAlertDialogToChosenServerInfo(
    serverName: ServerName,
    onPositiveButtonClick: () -> Unit = {},
    onNegativeButtonClick: () -> Unit = {},
) = generateConnectionConfirmationAlertDialogToChosenServerInfo(
    context = requireContext(),
    serverName = serverName,
    onPositiveButtonClick = onPositiveButtonClick,

```

```

        onNegativeButtonClick = onNegativeButtonClick
    )

fun generateConfirmationAlertDialog(
    context: Context,
    onContinueEditing: () -> Unit = {},
    onDiscardChanges: () -> Unit = {},
): AlertDialog = generateAlertDialog(
    context = context,
    title = context.getString(R.string.confirm_discard_changes),
    message =
context.getString(R.string.confirm_discard_youve_made_modifications),
    positiveButtonText = context.getString(R.string.continue_editing),
    onPositiveButtonClick = onContinueEditing,
    negativeButtonText = context.getString(R.string.just_discard_changes),
    onNegativeButtonClick = onDiscardChanges
)

fun Fragment.generateConfirmationAlertDialog(
    onContinueEditing: () -> Unit = {},
    onDiscardChanges: () -> Unit = {},
): AlertDialog {
    return generateConfirmationAlertDialog(requireContext(), onContinueEditing,
onDiscardChanges)
}

package com.edu.alterjuicechat.storage

import android.content.Context
import android.content.SharedPreferences
import com.edu.alterjuicechat.domain.storage.StorageStateFlow
import com.google.gson.Gson
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

abstract class Storage<TState>(
    private val stateType: Class<TState>,

```

```

): StorageStateFlow<TState> {
    private val gson = Gson()
    private val job = SupervisorJob()
    private val scope = CoroutineScope(job + Dispatchers.Default)
    protected abstract val preferencesFileName: String
    protected abstract val preferencesStateKey: String
    protected abstract val preferences: SharedPreferences
    abstract override val state: MutableStateFlow<TState>

    private fun getDefaultNewInstance(): TState { return stateType.newInstance()
}

fun invalidateAllData() = invalidateAllData(getDefaultNewInstance())
protected fun invalidateAllData(state: TState) { setProgress(state) }

protected fun callOnInit() = callOnInit(getDefaultNewInstance())
protected fun callOnInit(initialState: TState) {
    if (!preferences.getBoolean(KEY_IS_INITIALIZED, false)) {
        invalidateAllData(initialState)
        preferences.edit().putBoolean(KEY_IS_INITIALIZED, true).apply()
    }
    emitNewState(getProgress())
}

protected fun getProgress(): TState {
    return preferences.getString(preferencesStateKey, null)?.let {
        decode(it)
    }?: getDefaultNewInstance()
}

protected open fun setProgress(newState: TState){
    preferences.edit().putString(preferencesStateKey, newState?.let {
        encode(it)
    }).apply()
    emitNewState(newState)
}

protected fun transformProgress(transformer: (TState) -> TState){
    val progress = getProgress()
    val copy = transformer(progress)
    setProgress(copy)
}

```

```

        protected fun emitNewState(newValue: TState){
            scope.launch(Dispatchers.IO) {
                state.emit(newValue)
            }
        }

        protected fun getOrCreatePreferences(context: Context): SharedPreferences {
            return context.getSharedPreferences(preferencesFileName,
Context.MODE_PRIVATE)
        }

        private fun encode(progress: TState): String = gson.toJson(progress,
stateType)
        private fun decode(json: String): TState = gson.fromJson(json, stateType)

        companion object {
            private const val KEY_IS_INITIALIZED = "is_initialized"
        }
    }
}

```

```
package com.edu.alterjuicechat.storage
```

```

import android.content.Context
import android.content.SharedPreferences
import android.os.Build
import com.alterjuice.utils.isNullOrEmpty
import com.edu.alterjuicechat.domain.UserDisplayName
import com.edu.alterjuicechat.domain.storage.user_data.UserData
import com.edu.alterjuicechat.domain.storage.user_data.UserDataState
import kotlinx.coroutines.flow.MutableStateFlow

```

```

class UserDataStorage private constructor(context: Context):
Storage<UserDataState>(UserDataState::class.java), UserData {
    override val preferencesFileName = "com.edu.alterjuicechat.userdatastorage"
    override val preferencesStateKey= "userDataStorage"
    override val preferences: SharedPreferences =
getOrCreatePreferences(context)
    override val state = MutableStateFlow(getProgress())

```

```

    init { callOnInit(UserDataState(displayName = null.checkAndModifyIfEmpty()))
}

companion object {

    @Volatile
    private var INSTANCE: UserDataStorage? = null

    fun get(context: Context): UserDataStorage =
        INSTANCE ?: synchronized(this) {
            INSTANCE ?: UserDataStorage(context).also { INSTANCE = it }
        }

    override fun getDisplayName(): UserDisplayName {
        return state.value.displayName.checkAndModifyIfEmpty()
    }

    override fun setDisplayName(displayName: UserDisplayName) =
transformProgress {
        it.copy(displayName = displayName.checkAndModifyIfEmpty())
    }

    private fun UserDisplayName?.checkAndModifyIfEmpty(): UserDisplayName {
        return this.ifNullOrEmpty { Build.MODEL.ifNullOrEmpty { "User" } }
    }
}

package com.edu.alterjuicechat.storage

import android.content.Context
import android.content.SharedPreferences
import com.edu.alterjuicechat.domain.ServerUID
import
com.edu.alterjuicechat.domain.storage.session_data.dynamic.SessionDynamicData
import
com.edu.alterjuicechat.domain.storage.session_data.dynamic.SessionDynamicDataSta
te
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import kotlinx.coroutines.flow.*

```

```

class SessionDynamicDataStorage private constructor(
    context: Context,
    val serverUID: ServerUID
): Storage<SessionDynamicDataState>(SessionDynamicDataState::class.java),
SessionDynamicData {
    override val preferencesFileName: String = "${fileNamePrefix}_${serverUID}"
    override val preferencesStateKey: String = "sessionDynamicDataStorage"
    override val preferences: SharedPreferences =
        getOrCreatePreferences(context)
    override val state = MutableStateFlow(getProgress())

    init { callOnInit() }

    companion object {
        const val fileNamePrefix =
            "com.edu.alterjuicechat.sessiondynamicdatastorage"

        @Volatile
        private var INSTANCE: SessionDynamicDataStorage? = null

        private fun SessionDynamicDataStorage?.OrNull(serverUID: ServerUID):
            SessionDynamicDataStorage? {
            if (this == null)
                return null
            if (this.serverUID != serverUID)
                return null
            return this
        }

        /** Gets the saved session by ServerID.
         * So if user has already connected to server anytime -> he will get
         this session */
        fun get(context: Context, status: ConnectivityStatus.Connected):
            SessionDynamicDataStorage = get(context, status.serverUID)

        fun get(context: Context, serverUID: ServerUID):
            SessionDynamicDataStorage =
                INSTANCE.OrNull(serverUID) ?: synchronized(this) {
                    INSTANCE.OrNull(serverUID) ?: SessionDynamicDataStorage(context,
                        serverUID).also { INSTANCE = it }
                }
    }
}

```

```

        override fun transformState(block: (SessionDynamicDataState) ->
SessionDynamicDataState) {
            transformProgress(block)
        }

        override fun getSessionUID(): ServerUID? {
            return state.value.sessionUID
        }

        override fun setSessionUID(newSessionUID: ServerUID) = transformProgress {
            it.copy(sessionUID = newSessionUID)
        }

        override fun updateWithState(state: SessionDynamicDataState) {
            setProgress(state)
        }
    }

}

package com.edu.alterjuicechat.storage

import android.content.Context
import android.content.SharedPreferences
import com.edu.alterjuicechat.domain.ServerUID
import
com.edu.alterjuicechat.domain.storage.session_data.static.SessionStaticDataState
import
com.edu.alterjuicechat.domain.storage.session_data.static.SessionStaticData
import kotlinx.coroutines.flow.MutableStateFlow

class SessionStaticDataStorage private constructor(context: Context):
Storage<SessionStaticDataState>(
    SessionStaticDataState::class.java
), SessionStaticData {
    override val preferencesFileName: String =
"com.edu.alterjuicechat.sessionstaticdatastorage"
    override val preferencesStateKey: String = "sessionStaticDataStorage"
    override val preferences: SharedPreferences =
getOrCreatePreferences(context)
    override val state = MutableStateFlow(getProgress())

```

```

init { callOnInit() }

companion object {

    @Volatile
    private var INSTANCE: SessionStaticDataStorage? = null

    fun get(context: Context): SessionStaticDataStorage =
        INSTANCE ?: synchronized(this) {
            INSTANCE ?: SessionStaticDataStorage(context).also { INSTANCE =
it }
        }
    }

    override fun getSessionUID(): ServerUID {
        return state.value.sessionUID.orEmpty()
    }

    override fun setSessionUID(newSessionUID: ServerUID?) = transformProgress {
        it.copy(sessionUID = newSessionUID)
    }
}

package com.edu.alterjuicechat.storage

import android.content.Context
import android.os.Build
import com.alterjuice.utils.generateUID
import com.alterjuice.utils.isNullOrEmpty
import com.edu.alterjuicechat.domain.ServerName
import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.storage.server_data.HostData
import com.edu.alterjuicechat.domain.storage.server_data.HostDataState
import kotlinx.coroutines.flow.MutableStateFlow

class HostDataStorage private constructor(
    context: Context
): Storage<HostDataState>(HostDataState::class.java), HostData {
    override val preferencesFileName: String =
"com.edu.alterjuicechat.hoststaticdatastorage"
    override val preferencesStateKey: String = "hoststaticdatastorage"

```

```

override val preferences = getOrCreatePreferences(context)
override val state = MutableStateFlow(getProgress())

init { callOnInit(HostDataState(serverName = null.checkAndModifyIfEmpty()))
}

companion object {

    @Volatile
    private var INSTANCE: HostDataStorage? = null

    fun get(context: Context): HostDataStorage =
        INSTANCE ?: synchronized(this) {
            INSTANCE ?: HostDataStorage(context).also { INSTANCE = it }
        }
}

override fun revokeServerUID(): ServerUID {
    val newServerUID = generateUID()
    setServerUID(newServerUID)
    return newServerUID
}

override fun getServerUID(): ServerUID {
    return state.value.serverUID?: revokeServerUID()
}

override fun getServerName(): ServerName {
    return state.value.serverName.checkAndModifyIfEmpty()
}

override fun setServerUID(serverUID: String) = transformProgress {
    it.copy(serverUID = serverUID, allRegisteredUsers = mutableListOf())
}

override fun setServerName(serverName: String) = transformProgress {
    it.copy(serverName = serverName)
}

override fun checkIfUserNotRegisteredYet(userUID: UserUID): Boolean {
    return state.value.allRegisteredUsers.contains(userUID)
}

override fun registerIfUserNotRegisteredYet(userUID: UserUID){

```

```

        val it = getProgress()
        if (!it.allRegisteredUsers.contains(userID)){
            val copy = it.copy(allRegisteredUsers = it.allRegisteredUsers +
userUID)
            setProgress(copy)
        }
    }

    private fun ServerName?.checkAndModifyIfEmpty(): ServerName {
        return ifNullOrEmpty { Build.MODEL.ifNullOrEmpty { "User" } + "_Server"
    }
    }
}

package com.edu.alterjuicechat.extensions

import android.view.View

fun View.isVisibleOrGone(isVisible: Boolean) {
    if (isVisible) {
        this.makeVisible()
    } else {
        this.makeGone()
    }
}

fun View.makeInvisible() {
    this.visibility = View.INVISIBLE
}

fun View.makeVisible() {
    this.visibility = View.VISIBLE
}

fun View.makeGone() {
    this.visibility = View.GONE
}

package com.edu.alterjuicechat.extensions

import android.view.LayoutInflater
import android.view.ViewGroup

```

```

import androidx.appcompat.app.AppCompatActivity
import androidx.fragment.app.Fragment
import androidx.viewbinding.ViewBinding

inline fun <T : ViewBinding> Fragment.viewBinding(crossinline bindingInflater:
(LayoutInflater) -> T) =
    lazy(LazyThreadSafetyMode.NONE) {
        bindingInflater.invoke(layoutInflater)
    }

inline fun <T : ViewBinding> AppCompatActivity.viewBinding(crossinline
bindingInflater: (LayoutInflater) -> T) =
    lazy(LazyThreadSafetyMode.NONE) {
        bindingInflater.invoke(layoutInflater)
    }

private fun getInflater(container: ViewGroup?): LayoutInflater {
    return LayoutInflater.from(container?.context)
}

fun <T : ViewBinding> genBinding(
    bindingInflater: (LayoutInflater, ViewGroup?, Boolean) -> T,
    parent: ViewGroup?
) = genBinding(bindingInflater, parent, false)

fun <T : ViewBinding> genBinding(
    bindingInflater: (LayoutInflater, ViewGroup?, Boolean) -> T,
    parent: ViewGroup?,
    attachToParent: Boolean
) =
    bindingInflater.invoke(getInflater(parent), parent, attachToParent)

package com.edu.alterjuicechat.extensions

import com.edu.alterjuicechat.domain.NotificationIDs
import com.edu.alterjuicechat.notifications.model.NotificationMessage
import com.edu.alterjuicechat.server_model.dto.entities.MessageDto

```

```

fun MessageDto.toNotificationMessageWithID(): NotificationMessage {
    val msg = this
    return object : NotificationMessage{
        override val userName: String = msg.from.name
        override val contentText: String = msg.message
        override val chatID: String = msg.from.id
        override val notificationID: Int =
NotificationIDs.MESSAGE_NOTIFICATION_ID++
    }
}

package com.edu.alterjuicechat.extensions

import android.graphics.Paint
import android.graphics.drawable.AnimationDrawable
import android.view.View
import android.view.animation.AnimationUtils
import android.widget.Button
import android.widget.TextView
import com.edu.alterjuicechat.R

fun View.scaleUpDown(increaseBy: Float = 1.1f, duration: Long = 200){

    this.animate().scaleY(increaseBy).scaleX(increaseBy).setDuration(duration).withE
ndAction{
        this.animate().scaleX(1f).scaleY(1f).start()
    }
}

fun View.animateBackgroundGradient(){
    (this.background as? AnimationDrawable)?.let {
        it.setEnterFadeDuration(2000)
        it.setExitFadeDuration(4000)
        it.start()
    }
}

fun View.animateWithScaleXYInfinately(){
    val scaleUpDownAnimation = AnimationUtils.loadAnimation(this.context,
R.anim.view_scale_xy_up_down)
    this.animation?.cancel()
    this.startAnimation(scaleUpDownAnimation)
}

```

```
fun TextView.makeTextUnderlined() {
    this.paintFlags = this.paintFlags or Paint.UNDERLINE_TEXT_FLAG
}
```

```
fun Button.makeButtonAvailableToManageService(text: String,
blockToStartOrStopService: () -> Unit){
    this.animate().cancel()
    this.isEnabled = true
    this.text = text
    this.setOnClickListener {
        this.animate().withStartAction{ isEnabled = false }
            .setDuration(5000L).withEndAction { isEnabled = true }
            .start()
        blockToStartOrStopService()
    }
}
```

```
fun View.animateExtended(
    slideFromX: Float = 0f, slideToX: Float = 0f,
    slideFromY: Float = 0f, slideToY: Float = 0f,
    fromScaleX: Float = 1f, toScaleX: Float = 1f,
    fromScaleY: Float = 1f, toScaleY: Float = 1f,
    fromAlpha: Float = 1f, toAlpha: Float = 1f,
    duration: Long = 1_000,
    startDelay: Long = 0L,
    withStartAction: () -> Unit = {},
    withEndAction: () -> Unit = {},

) {
    with(this){
        alpha = fromAlpha
        translationY = slideFromY
        translationX = slideFromX
        scaleX = fromScaleX
        scaleY = fromScaleY
    }
    this.animate()
        .withStartAction(withStartAction)
        .translationX(slideToX)
        .translationY(slideToY)
        .scaleX(toScaleX)
        .scaleY(toScaleY)
        .alpha(toAlpha)
```

```

        .setDuration(duration)
        .setStartDelay(startDelay)
        .withEndAction(withEndAction)

    }

package com.edu.alterjuicechat.extensions

import android.app.NotificationManager
import android.app.Service
import android.content.Context
import android.content.Intent
import com.edu.alterjuicechat.services.BaseForegroundService
import com.edu.alterjuicechat.services.getIntentToStopService

inline fun <reified T: Service> startMyForegroundService(context: Context?) {
    context?.startForegroundService(Intent(context, T::class.java))
}

inline fun <reified T: BaseForegroundService> stopMyForegroundService(context:
Context){
    val intent = getIntentToStopService<T>(context)
    context.startService(intent)
}

fun Context.getSystemNotificationManager(): NotificationManager {
    return this.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
}

package com.edu.alterjuicechat.repo

import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.model.Chat
import com.edu.alterjuicechat.domain.model.Message
import kotlinx.coroutines.flow.MutableStateFlow

object GlobalFlows {

    val allChats = MutableStateFlow<List<Chat>>(emptyList())

```

```

        val activeChatUID = MutableStateFlow<UserUID?>(null)
        val activeChat = MutableStateFlow<Chat?>(null)
        val activeChatMessages = MutableStateFlow<List<Message>>(emptyList())
    }

```

```
package com.edu.alterjuicechat.repo
```

```

import com.edu.alterjuicechat.domain.model.User
import com.edu.alterjuicechat.server_model.dto.entities.BaseDto

```

```

interface IClientWorker {
    fun onNewPayload(payload: String)
    fun sendToServer(action: BaseDto)
    suspend fun requestUsers()
    fun recreateUsersRequester()
    fun sendMessage(receiver: User, message: String)
    suspend fun tryToConnect()
    suspend fun disconnect(code: Int)
    suspend fun stopAll()
}

```

```
package com.edu.alterjuicechat.repo
```

```

import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.flow.StateFlow

```

```

abstract class RuntimeStatusHolder<T> {
    protected val job = SupervisorJob()
    protected val scope = CoroutineScope(Dispatchers.Default + job)
    protected var selfTurnerOff: Job? = null

    abstract val statusState: StateFlow<T>
    abstract fun emitNewStatus(newStatus: T)
    abstract fun notifySelfPinged()
    protected abstract fun recreateSelfTurnerOff()
}

```

```
package com.edu.alterjuicechat.repo
```

```

import com.alterjuice.utils.HandlerHolder
import com.alterjuice.utils.RenewableJob
import com.edu.alterjuice.socket_server.JClient
import com.edu.alterjuicechat.domain.ServerConsts
import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.model.Chat
import com.edu.alterjuicechat.domain.model.LastMessage
import com.edu.alterjuicechat.domain.model.Message
import com.edu.alterjuicechat.domain.model.User
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import com.edu.alterjuicechat.domain.storage.user_data.UserData
import com.edu.alterjuicechat.server_model.dto.GeneratorDto
import com.edu.alterjuicechat.server_model.dto.ParserDto
import com.edu.alterjuicechat.server_model.dto.entities.BaseDto
import com.edu.alterjuicechat.server_model.dto.entities.ConnectedDto
import com.edu.alterjuicechat.server_model.dto.entities.MessageDto
import com.edu.alterjuicechat.server_model.dto.entities.UserDto
import kotlinx.coroutines.*
import org.koin.core.component.KoinComponent
import kotlin.collections.List
import kotlin.collections.forEach
import kotlin.collections.isNotEmpty
import kotlin.collections.mutableMapOf
import kotlin.collections.set
import kotlin.random.Random

class ClientWorker constructor(
    private val parser: ParserDto,
    private val generator: GeneratorDto,
    private val userData: UserData,
) : IClientWorker, IDataRepo,
    HandlerHolder<(UserUID?, MessageDto) -> Unit> {

    private val client = JClient(this::onNewPayload)

    private val dataRepo get() = RuntimeConnectivity.dataRepoState.value

    private val messageHandlers = mutableMapOf<String, (UserUID?, MessageDto) ->
Unit>()

    private var isTriedToAuthAs = false

```

```

private val job = SupervisorJob()
private val scope = CoroutineScope(Dispatchers.Default + job)

private val pingPongJob = RenewableJob(::launchPingPong)
private val tryConnectOnInitJob = RenewableJob(::generateTryToConnectJob)
private val autoUsersRequesterJob = RenewableJob(::generateUsersUpdaterJob)

private fun generateUsersUpdaterJob() = scope.launch {
    while (true){
        requestUsers()
        delay(5000L)
    }
}

override suspend fun tryToConnect() {
    isTriedToAuthAs = false
    tryConnectOnInitJob.renew()
}

private fun generateTryToConnectJob() = scope.launch {
    while (true) {
        val status = RuntimeConnectivity.statusState.value
        if (status is ConnectivityStatus.ChosenServerToConnect) {
            client.initialize(scope, status.ip, ServerConsts.SERVER_PORT)
            // repoWrapper.renew()
            break
            // We are launching events listener on client initializing
        }
        delay(1000)
    }
    stopTryToConnectJob()
}

private fun stopTryToConnectJob() {
    tryConnectOnInitJob.cancelIt()
}

override fun onNewPayload(payload: String) {
    scope.launch {
        val baseDto = parser.parseBaseDto(payload)
    }
}

```

```

        handleUpdate(baseDto)
    }
}

private suspend fun handleUpdate(baseDto: BaseDto) {
    when (baseDto.action) {
        BaseDto.Action.PONG -> {
            val pongDto = parser.parsePong(baseDto.payload)
            withContext(Dispatchers.Main) {
                RuntimeConnectivity.notifySelfPinged()
            }
        }
        BaseDto.Action.USERS_RECEIVED -> {
            val usersDto = parser.parseUsersReceived(baseDto.payload)
            dataRepo.handleNewUsers(usersDto.users)
        }
        BaseDto.Action.CONNECTED -> {
            onActionConnected(parser.parseConnected(baseDto.payload))
        }
        BaseDto.Action.NEW_MESSAGE -> {
            onActionNewMessage(parser.parseMessage(baseDto.payload))
        }
        else -> { /* Nothing to do */
        }
    }
}

private suspend fun onActionConnected(connectedDto: ConnectedDto) {

    if (!isTriedToAuthAs) {
        val status = RuntimeConnectivity.statusState.value
        if (status is ConnectivityStatus.ChosenServerToConnect) {
            RuntimeConnectivity.emitNewStatus(status.toConnectedToServer())
        }
        var i = 0
        while (dataRepo.getDynamicSessionUID() == null) {
            delay(300)
            if (i == 3)
                break /* So there is no sessions before. connecting as it
is*/
            i++
        }
    }
}

```

```

        val initSessionID = connectedDto.id
        /* oldSessionID -> an old userSessionUID which was given from the
last connection
        * to the same serverUID */
        val oldSessionID = dataRepo.getDynamicSessionUID()
        if (oldSessionID.isNullOrEmpty()) {
            // was not any sessions before
        } else {
            val action = generator.generateAuthAs(initSessionID,
oldSessionID)

            sendToServer(action)
            isTriedToAuthAs = true
            return
        }
    }
    val finalSessionID = connectedDto.id

    println("TCP Connected! SessionID: ${finalSessionID};")
    connect(finalSessionID)
    val status = RuntimeConnectivity.statusState.value
    if (status is ConnectivityStatus.Connected.ConnectedToServer) {

RuntimeConnectivity.emitNewStatus(status.toAuthenticated(finalSessionID))
        withContext(Dispatchers.Main) {
            dataRepo.setDynamicSessionUID(finalSessionID)
        }
    }

    pingPongJob.renew()
    autoUsersRequesterJob.renew()
}

private suspend fun onActionNewMessage(newMessageDto: MessageDto) {
    if (messageHandlers.isNotEmpty()) {
        messageHandlers.forEach {
it.value.invoke(GlobalFlows.activeChatUID.value, newMessageDto) }
    }
    dataRepo.handleReceiveMessage(newMessageDto)
}

private fun launchPingPong() = scope.launch(Dispatchers.IO) {
    while (true) {

```

```

        workWithSessionID { finalSessionID ->
            sendToServer(generator.generatePing(finalSessionID))
        }
        delay(Random.nextLong(2000, 4500))
        // Random time to wait
    }
}

override fun recreateUsersRequester() {
    autoUsersRequesterJob.renew()
}

override suspend fun requestUsers() = workWithSessionID {
    val action = generator.generateGetUsers(it)
    sendToServer(action)
}

override fun sendMessage(receiver: User, message: String) =
workWithSessionID {
    val action = generator.generateSendMessage(it, receiver.userID, message)
    sendToServer(action)
    dataRepo.handleSendMessage(it, receiver, message)
}

override fun sendToServer(action: BaseDto) {
    try {
        client.send(generator.toJson(action))
    } catch (e: Exception) {
        e.printStackTrace()
    }
}

private fun connect(finalSessionID: String) {
    // FIRST STEP TO CONNECT
    val username = userData.getDisplayName()
    val action = generator.generateConnect(finalSessionID, username)
    sendToServer(action)
}

override suspend fun disconnect(code: Int) = workWithSessionID {
    val action = generator.generateDisconnect(it, code)
    sendToServer(action)
}

```

```

    override suspend fun stopAll() {
        isTriedToAuthAs = false
        pingPongJob.cancelIt()
        tryConnectOnInitJob.cancelIt()
        autoUsersRequesterJob.cancelIt()
        client.awaitAndClose()
        messageHandlers.clear()
    }

    private inline fun workWithSessionID(block: (sessionId: String) -> Unit) {
        (RuntimeConnectivity.statusState.value as?
ConnectivityStatus.Connected.Authenticated)?.finalSessionID?.let(block)
    }

    override fun getMessages(userID: UserID): List<Message> =
dataRepo.getMessages(userID)
    override fun getChat(userID: UserID): Chat = dataRepo.getChat(userID)
    override fun getLastMessage(userID: UserID): LastMessage? =
dataRepo.getLastMessage(userID)

    override fun handleSendMessage(sessionID: UserID, receiver: User, message:
String) =
        dataRepo.handleSendMessage(sessionID, receiver, message)

    override fun handleReceiveMessage(messageDto: MessageDto) =
        dataRepo.handleReceiveMessage(messageDto)

    override fun handleMessage(userID: UserID, messageDto: MessageDto,
increaseCounter: Boolean) =
        dataRepo.handleMessage(userID, messageDto, increaseCounter)

    override fun handleNewUsers(newUsers: List<UserDto>) =
dataRepo.handleNewUsers(newUsers)
    override fun setActiveChat(userID: UserID?) =
dataRepo.setActiveChat(userID)
    override fun updateFlows() = dataRepo.updateFlows()
    override fun updateMessagesFlows() = dataRepo.updateMessagesFlows()
    override fun clearUnreadCounter(userID: UserID) =
dataRepo.clearUnreadCounter(userID)
    override fun getDynamicSessionUID(): UserID? =
dataRepo.getDynamicSessionUID()

```

```

        override fun setDynamicSessionUID(sessionUID: UserID) =
dataRepo.setDynamicSessionUID(sessionUID)
        override fun addHandler(key: String, handler: (UserID?, MessageDto) ->
Unit) {
            messageHandlers[key] = handler
        }
    }

}

package com.edu.alterjuicechat.repo

import com.edu.alterjuicechat.domain.UserID
import com.edu.alterjuicechat.domain.model.*
import com.edu.alterjuicechat.server_model.dto.entities.MessageDto
import com.edu.alterjuicechat.server_model.dto.entities.UserDto

val emptyIRepo = object : IDataRepo {
    override fun getMessages(userID: UserID): List<Message> = emptyList()
    override fun getChat(userID: UserID): Chat { return DataChat(userID, "") }
}

    override fun getLastMessage(userID: UserID): LastMessage? = null
    override fun handleSendMessage(sessionID: UserID, receiver: User, message:
String) {}
    override fun handleReceiveMessage(messageDto: MessageDto) {}
    override fun handleMessage(userID: UserID, messageDto: MessageDto,
increaseCounter: Boolean) {}
    override fun handleNewUsers(newUsers: List<UserDto>) {}
    override fun setActiveChat(userID: UserID?) {}
    override fun updateFlows() {}
    override fun updateMessagesFlows() {}
    override fun clearUnreadCounter(userID: UserID) {}
    override fun getDynamicSessionUID(): UserID? = null
    override fun setDynamicSessionUID(sessionUID: UserID) {}
}

package com.edu.alterjuicechat.repo

import com.edu.alterjuicechat.domain.model.HostStatus
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.MutableStateFlow

object RuntimeHost: RuntimeStatusHolder<HostStatus>() {

```

```

override val statusState = MutableStateFlow<HostStatus>(HostStatus.Disabled)

override fun emitNewStatus(newStatus: HostStatus) {
    scope.launch {
        statusState.emit(newStatus)
        if (newStatus is HostStatus.Broadcasting){
            recreateSelfTurnerOff()
        }
    }
}

override fun notifySelfPinged() {
    recreateSelfTurnerOff()
}

override fun recreateSelfTurnerOff(){
    selfTurnerOff?.cancel()
    selfTurnerOff = scope.launch {
        delay(10_000)
        if (statusState.value is HostStatus.Broadcasting) {
            statusState.emit(HostStatus.Disabled)
        }
    }
}
}

```

```
package com.edu.alterjuicechat.repo
```

```

import android.content.Context
import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.domain.UserDisplayName
import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.model.Chat
import com.edu.alterjuicechat.domain.model.LastMessage
import com.edu.alterjuicechat.domain.model.Message
import com.edu.alterjuicechat.domain.model.User
import
com.edu.alterjuicechat.domain.storage.session_data.dynamic.SessionDynamicDataSta
te
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import com.edu.alterjuicechat.server_model.dto.entities.MessageDto
import com.edu.alterjuicechat.server_model.dto.entities.UserDto
import com.edu.alterjuicechat.storage.SessionDynamicDataStorage

```

```

import com.edu.alterjuicechat.storage.SessionStaticDataStorage
import kotlinx.coroutines.*

class DataRepo private constructor(
    private val userDisplayName: UserDisplayName,
    private val serverUID: ServerUID,
    private val dynamicSessionData: SessionDynamicDataStorage,
): IDataRepo {

    companion object {

        @Volatile
        private var INSTANCE: DataRepo? = null

        private fun DataRepo?.OrNull(serverUID: ServerUID): DataRepo? {
            if (this == null)
                return null
            if (this.serverUID != serverUID)
                return null
            return this
        }

        /** Gets the saved session by ServerID.
         * So if user has already connected to server anytime -> he will get
         this session */
        fun get(context: Context, status: ConnectivityStatus.Connected,
            userDisplayName: UserDisplayName): DataRepo =
            get(context, status.serverUID, userDisplayName)

        fun get(context: Context, serverUID: ServerUID, userDisplayName:
            UserDisplayName): DataRepo =
            INSTANCE.OrNull(serverUID) ?: synchronized(this) {
                INSTANCE.OrNull(serverUID) ?: DataRepo(userDisplayName,
                    serverUID, SessionDynamicDataStorage.get(context, serverUID)).also { INSTANCE =
                    it }
            }

        fun create(context: Context, status: ConnectivityStatus.Connected,
            userDisplayName: UserDisplayName){
            create(context, status.serverUID, userDisplayName)
        }
    }
}

```

```

        fun create(context: Context, serverUID: ServerUID, userDisplayName:
UserDisplayName){
            INSTANCE.orNull(serverUID) ?: synchronized(this) {
                INSTANCE.orNull(serverUID) ?: DataRepo(userDisplayName,
serverUID, SessionDynamicDataStorage.get(context, serverUID)).also { INSTANCE =
it }
            }
        }

        fun clear(){
            synchronized(this){ INSTANCE = null }
        }
    }

    private val job = SupervisorJob()
    private val scope = CoroutineScope(job + Dispatchers.Default)
    private var updateFlowJob: Job? = null

    init {
        updateFlows()
    }

    override fun getMessages(userUID: UserUID): List<Message> {
        return dynamicSessionData.state.value.getMessages(userUID)
    }

    override fun getChat(userUID: UserUID): Chat {
        return dynamicSessionData.state.value.getChat(userUID)
    }

    override fun getLastMessage(userUID: UserUID): LastMessage? {
        return dynamicSessionData.state.value.getLastMessage(userUID)
    }

    override fun handleSendMessage(sessionID: UserUID, receiver: User, message:
String){
        dynamicSessionData.state.value.createUserIfNeeded(receiver.userID,
receiver.userName)
        dynamicSessionData.updateWithState(dynamicSessionData.state.value)
    }

```

```

        val messageDto = MessageDto(from=UserDto(sessionID, userDisplayName),
message)
        handleMessage(receiver.userID, messageDto, false)
    }

    override fun handleReceiveMessage(messageDto: MessageDto) {
        dynamicSessionData.transformState {
            it.createUserIfNeeded(messageDto.from.id, messageDto.from.name)
            it
        }
        val chatID = messageDto.from.id
        handleMessage(chatID, messageDto, true)
    }

    override fun handleMessage(userID: UserID, messageDto: MessageDto,
increaseCounter: Boolean){
        addMessage(userID, messageDto, increaseCounter)
        updateFlows()
        updateMessagesFlows()
    }

    private fun addMessage(userID: UserID, message: MessageDto,
increaseCounter: Boolean) {
        dynamicSessionData.transformState {
            it.addMessage(userID, message, increaseCounter)
            it
        }
    }

    override fun handleNewUsers(newUsers: List<UserDto>) {
        dynamicSessionData.state.value.handleNewUsers(newUsers)
        dynamicSessionData.updateWithState(dynamicSessionData.state.value)
        updateFlows()
    }

    override fun setActiveChat(userID: UserID?){
        scope.launch(Dispatchers.Main) {
            GlobalFlows.activeChatUID.emit(userID)
            if (userID == null) {
                GlobalFlows.activeChat.emit(null)
                GlobalFlows.activeChatMessages.emit(emptyList())
            } else {
                clearUnreadCounter(userID)
            }
        }
    }

```

```

        GlobalFlows.activeChatMessages.emit(getMessages(userUID))
        GlobalFlows.activeChat.emit(getChat(userUID))
    }
}

override fun updateFlows() {
    updateFlowJob?.cancel()
    updateFlowJob = scope.launch {
        withContext(Dispatchers.Main) {
            val chats = dynamicSessionData.state.value.getChats()
            GlobalFlows.allChats.emit(chats)
        }
    }
}

override fun updateMessagesFlows() {
    setActiveChat(GlobalFlows.activeChatUID.value)
}

override fun clearUnreadCounter(userUID: UserUID) {
    dynamicSessionData.transformState {
        it.clearUnreadCounter(userUID)
        it
    }
}

override fun getDynamicSessionUID(): UserUID? {
    return dynamicSessionData.getSessionUID()
}

override fun setDynamicSessionUID(sessionUID: UserUID) {
    dynamicSessionData.setSessionUID(sessionUID)
}
}

package com.edu.alterjuicechat.repo

import android.content.Context
import android.os.Build
import com.alterjuice.utils.RenewableJob

```

```

import
com.edu.alterjuicechat.domain.storage.session_data.dynamic.SessionDynamicData
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import
com.edu.alterjuicechat.domain.storage.session_data.static.SessionStaticData
import com.edu.alterjuicechat.domain.storage.user_data.UserData
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.collect
import org.koin.core.component.KoinComponent
import org.koin.core.component.get
import org.koin.core.parameter.parametersOf

object RuntimeConnectivity: RuntimeStatusHolder<ConnectivityStatus>(),
KoinComponent {
    val dataRepoState = MutableStateFlow<IDataRepo>(emptyIREpo)

    private val additionalCollectorJob = RenewableJob { scope.launch {
        statusState.collect { status ->
            if (status is ConnectivityStatus.Connected) {
                if (status is ConnectivityStatus.Connected.Authenticated){
                    withContext(Dispatchers.Main) {

get<SessionStaticData>().setSessionUID(status.finalSessionID)
                        get<SessionDynamicData>(){ parametersOf(status)
                    }.setSessionUID(status.finalSessionID)
                        }
                    }
                withContext(Dispatchers.Main) {
                    val newDataRepo = DataRepo.get(get<Context>(), status,
get<UserData>().getDisplayName())
                        dataRepoState.emit(newDataRepo)

                    }
                } else if (status is ConnectivityStatus.NotConnected){
                    withContext(Dispatchers.Main){
                        get<SessionStaticData>().setSessionUID(null)
                        dataRepoState.emit(emptyIREpo)
                    }
                }
            }
        }
    }
}
}

```

```

init { additionalCollectorJob.renew() }

override val statusState =
MutableStateFlow<ConnectivityStatus>(ConnectivityStatus.NotConnected)

override fun emitNewStatus(newStatus: ConnectivityStatus) {
    scope.launch {
        statusState.emit(newStatus)
        if (newStatus is ConnectivityStatus.Connected){
            recreateSelfTurnerOff()
        }
    }
}

override fun notifySelfPinged() {
    recreateSelfTurnerOff()
}

override fun recreateSelfTurnerOff(){
    selfTurnerOff?.cancel()
    selfTurnerOff = scope.launch {
        println("${Build.MODEL} NOTIFIED_PONGED")

        delay(10_000)
        if (statusState.value is ConnectivityStatus.Connected.Authenticated)
{
            statusState.emit(ConnectivityStatus.NotConnected)
            withContext(Dispatchers.Main){
                println("${Build.MODEL}
DATA_REPO_CLEARED*****")
                DataRepo.clear()
            }
        }
    }
}

}

package com.edu.alterjuicechat.repo

import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.model.Chat
import com.edu.alterjuicechat.domain.model.LastMessage

```

```

import com.edu.alterjuicechat.domain.model.Message
import com.edu.alterjuicechat.domain.model.User
import com.edu.alterjuicechat.server_model.dto.entities.MessageDto
import com.edu.alterjuicechat.server_model.dto.entities.UserDto

interface IDataRepo {

    fun getMessages(userID: UserID): List<Message>

    fun getChat(userID: UserID): Chat
    fun getLastMessage(userID: UserID): LastMessage?
    fun handleSendMessage(sessionID: UserID, receiver: User, message: String)
    fun handleReceiveMessage(messageDto: MessageDto)
    fun handleMessage(userID: UserID, messageDto: MessageDto, increaseCounter:
Boolean)
    fun handleNewUsers(newUsers: List<UserDto>)
    fun setActiveChat(userID: UserID?)
    fun updateFlows()
    fun updateMessagesFlows()
    fun clearUnreadCounter(userID: UserID)
    fun getDynamicSessionUID(): UserID?
    fun setDynamicSessionUID(sessionUID: UserID)

}

```

```

package com.edu.alterjuicechat.notifications

```

```

import android.app.NotificationChannel
import com.edu.alterjuicechat.notifications.data.JNotifications
import com.edu.alterjuicechat.notifications.model.JNotification

fun JNotification.getAsChannel(): NotificationChannel {
    return NotificationChannel(this.channelID, this.channelName,
this.importance).also {
        it.description = this.channelDescription
        it.lockscreenVisibility = this.notificationVisibility
    }
}

fun JNotifications.getAsChannels(): List<NotificationChannel> {

```

```

        return this.values.map { it.getAsChannel() }
    }

package com.edu.alterjuicechat.notifications.notification_creator

import android.app.Notification
import android.content.Context
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.domain.NotificationIDs
import com.edu.alterjuicechat.domain.ServerIP
import com.edu.alterjuicechat.domain.ServerName
import com.edu.alterjuicechat.extensions.getSystemNotificationManager
import
com.edu.alterjuicechat.navigation.generatePendingIntentForHostSettingsDestinatio
n
import com.edu.alterjuicechat.notifications.*
import com.edu.alterjuicechat.notifications.data.JNotifications

object BroadcastingNotificationGenerator {
    fun createNotification(
        context: Context,
        serverIP: ServerIP,
        serverName: ServerName
    ): Notification.Builder {

        return createNotificationWithoutText(context)

        .setContentTitle(context.getString(R.string.notification_server_is_running_on,
serverIP))

        .setContentText(context.getString(R.string.notification_click_to_operate_the_ser
ver))
    }

    fun createAndSendNotification(
        context: Context,
        serverIP: ServerIP,
        serverName: ServerName
    ) {
        val notificationManager = context.getSystemNotificationManager()
        checkNotificationChannels(notificationManager)
    }
}

```

```

        val notificationBuilder = createNotification(context, serverIP,
serverName)
        notificationManager.notify(NotificationIDs.SERVER_NOTIFICATION_ID,
notificationBuilder.build())
    }

    fun createNotificationWithoutText(context: Context): Notification.Builder {
        val notificationConfig = JNotifications.broadcasting
        return interactNotificationWithDefaultValues(context,
notificationConfig)

.setContentIntent(generatePendingIntentForHostSettingsDestination(context))

    }

}

package com.edu.alterjuicechat.notifications.notification_creator

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.content.Context
import com.alterjuice.utils.contains
import com.alterjuice.utils.tryCatch
import com.edu.alterjuicechat.extensions.getSystemNotificationManager
import com.edu.alterjuicechat.notifications.data.JNotifications
import com.edu.alterjuicechat.notifications.getAsChannels
import com.edu.alterjuicechat.notifications.model.JNotification

fun interactNotificationWithDefaultValues(
    context: Context,
    notificationConfig: JNotification
): Notification.Builder {
    notificationConfig.channelID
    return Notification.Builder(context, notificationConfig.channelID)
        .setSmallIcon(notificationConfig.smallIconResID)
        .setCategory(notificationConfig.notificationCategory)
        .setOngoing(!notificationConfig.isRemovable)
        .setWhen(0)
        .setAutoCancel(false)
}

```

```

internal fun checkNotificationChannels(context: Context){
    return checkNotificationChannels(
        notificationManager = context.getSystemService(),
        channels = JNotifications.getAsChannels()
    )
}

internal fun checkNotificationChannels(notificationManager:
NotificationManager){
    return checkNotificationChannels(
        notificationManager = notificationManager,
        channels = JNotifications.getAsChannels()
    )
}

internal fun checkNotificationChannels(
    notificationManager: NotificationManager,
    channels: List<NotificationChannel>
) {
    val installedChannels = notificationManager.notificationChannels
    for (supportedChannel in channels){
        if (!installedChannels.contains { it.id == supportedChannel.id }){
            tryCatch("Channel ${supportedChannel.id}") {
                notificationManager.createNotificationChannel(supportedChannel)
            }
        }
    }
}

fun cancelNotification(context: Context, notificationID: Int) {
    val notificationManager= context.getSystemService()
    notificationManager.cancel(notificationID)
}

package com.edu.alterjuicechat.notifications.notification_creator

import android.app.Notification
import android.content.Context
import android.graphics.drawable.Icon
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.domain.NotificationIDs
import com.edu.alterjuicechat.extensions.getSystemService

```

```

import
com.edu.alterjuicechat.navigation.generatePendingIntentToStopScanningServers
import
com.edu.alterjuicechat.navigation.generatePendingIntentToViewScannedServersDesti
nation
import com.edu.alterjuicechat.notifications.data.JNotifications

object ServersScannerNotificationGenerator {
    fun createNotification(context: Context, countAvailableServers: Int):
Notification.Builder {
        val notificationConfig = JNotifications.serversScanner
        return interactNotificationWithDefaultValues(context,
notificationConfig)

        .setContentTitle(context.getString(R.string.notification_searching_for_servers_f
ound_x, countAvailableServers))

        .setContentText(context.getString(R.string.notification_click_to_view_list_of_se
rvers))

        .addAction(Notification.Action.Builder(
            Icon.createWithResource(context, R.drawable.ic_round_search_24),
            context.getString(R.string.notification_view_servers),
            generatePendingIntentToViewScannedServersDestination(context)
        ).build())
        .addAction(Notification.Action.Builder(
            Icon.createWithResource(context, R.drawable.ic_round_search_24),
            context.getString(R.string.notification_stop_listennning),
            generatePendingIntentToStopScanningServers(context)
        ).build())

        .setContentIntent(generatePendingIntentToViewScannedServersDestination(context))
    }

    fun createAndSendNotification(context: Context, countAvailableServers: Int)
{
        val notificationManager = context.getSystemService(NotificationManager())
        checkNotificationChannels(notificationManager)
        val notificationBuilder = createNotification(context,
countAvailableServers)

        notificationManager.notify(NotificationIDs.SEARCHING_SERVERS_NOTIFICATION_ID,
notificationBuilder.build())
    }

```

```
}
```

```
package com.edu.alterjuicechat.notifications.notification_creator
```

```
import android.app.Notification
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.domain.NotificationIDs
import com.edu.alterjuicechat.domain.ServerIP
import com.edu.alterjuicechat.domain.ServerName
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import com.edu.alterjuicechat.extensions.getSystemNotificationManager
import com.edu.alterjuicechat.notifications.data.JNotifications
import com.edu.alterjuicechat.ui.MainActivity
```

```
object ConnectedToServerNotificationGenerator {
    fun createNotification(context: Context, serverIP: ServerIP, serverName:
ServerName): Notification.Builder {
        val notificationConfig = JNotifications.connectedToServer
        return interactNotificationWithDefaultValues(context,
notificationConfig)

        .setContentTitle(context.getString(R.string.notification_not_connected))
    }

    fun createNotificationWithStatus(
        context: Context,
        status: ConnectivityStatus
    ) : Notification.Builder{
        val notificationConfig = JNotifications.connectedToServer
        return interactNotificationWithDefaultValues(context,
notificationConfig)
        .apply {
            when (status){
                is ConnectivityStatus.ChosenServerToConnect -> {
                    setContentTitle("Trying to connect to ${status.ip}")
                }
                is ConnectivityStatus.Connected.Authenticated -> {
                    setContentTitle("Authenticated to server on
${status.ip}")
                }
            }
        }
    }
}
```

```

        }
        is ConnectivityStatus.Connected.ConnectedToServer -> {
            setContentTitle("Connected to server on ${status.ip}")
        }
        ConnectivityStatus.NotConnected -> {

setContentTitle(context.getString(R.string.notification_not_connected))
        }
    }
    if (status is ConnectivityStatus.Connected){

setContentText(context.getString(R.string.notification_click_to_view_all_chats))
        setContentIntent(PendingIntent.getActivity(context, 0,
Intent(context, MainActivity::class.java), PendingIntent.FLAG_UPDATE_CURRENT or
PendingIntent.FLAG_IMMUTABLE))
        }
    }
}

fun createAndSendNotification(context: Context, serverIP: ServerIP,
serverName: ServerName) {
    val notificationManager = context.getSystemService()
    val notification = createNotification(context, serverIP, serverName)
    checkNotificationChannels(context)
    notificationManager.notify(NotificationIDs.SERVER_NOTIFICATION_ID,
notification.build())
}

}

package com.edu.alterjuicechat.notifications.notification_creator

import android.app.Notification
import android.content.Context
import com.edu.alterjuicechat.extensions.getSystemService
import
com.edu.alterjuicechat.navigation.generatePendingIntentForDirectChatDestination
import com.edu.alterjuicechat.notifications.data.JNotifications
import com.edu.alterjuicechat.notifications.model.JNotification
import com.edu.alterjuicechat.notifications.model.NotificationMessage

object MessageNotificationGenerator {

```

```

fun createNotification(
    context: Context,
    configuration: JNotification,
    message: NotificationMessage,
): Notification.Builder {
    return interactNotificationWithDefaultValues(context, configuration)
        .setContentTitle(message.userName)
        .setContentText(message.contentText)
        .setContentIntent(
            generatePendingIntentForDirectChatDestination(context,
message.chatID, message.userName)
        )
}

```

```

fun createAndSendNotification(context: Context, message:
NotificationMessage) {
    val notificationManager = context.getSystemService()
    checkNotificationChannels(notificationManager)

    val notificationBuilder: Notification.Builder = createNotification(
        context = context,
        configuration = JNotifications.messages,
        message = message
    )
    notificationManager.notify(message.notificationID,
notificationBuilder.build())
}
}

```

```

package com.edu.alterjuicechat.notifications.model

```

```

interface DeclaredNotificationTypes<T> {
    val broadcasting: T
    val messages: T
    val serversScanner: T
    val connectedToServer: T
}

```

```

val values: List<T> get() = listOf(
    broadcasting,
    messages,
    serversScanner,
    connectedToServer,
)

```

```
)
}
```

```
package com.edu.alterjuicechat.notifications.model
```

```
interface NotificationMessage {
    val userName: String
    val contentText: String
    val chatID: String
    val notificationID: Int
}
```

```
package com.edu.alterjuicechat.notifications.model
```

```
interface JNotification {
    val channelID: String
    val channelName: String
    val importance: Int
    val notificationVisibility: Int
    val channelDescription: String get() = channelName

    val smallIconResID: Int
    val notificationCategory: String
    val isRemovable: Boolean
    val pendingIntentRequestCode: Int get() = 112
}
```

```
data class JNotificationData(
    override val channelID: String,
    override val channelName: String,
    override val importance: Int,
    override val notificationVisibility: Int,
    override val channelDescription: String = channelName,
    override val smallIconResID: Int,
    override val notificationCategory: String,
    override val isRemovable: Boolean,
    override val pendingIntentRequestCode: Int = 112,
): JNotification
```

```
package com.edu.alterjuicechat.notifications.data
```

```

import android.app.Notification
import android.app.NotificationManager
import com.edu.alterjuicechat.R
import com.edu.alterjuicechat.notifications.model.DeclaredNotificationTypes
import com.edu.alterjuicechat.notifications.model.JNotification
import com.edu.alterjuicechat.notifications.model.JNotificationData

object JNotifications: DeclaredNotificationTypes<JNotification>{
    override val broadcasting = JNotificationData(
        channelID = "Server",
        channelName = "Server notification",
        importance = NotificationManager.IMPORTANCE_MAX,
        notificationVisibility = Notification.VISIBILITY_PUBLIC,
        smallIconResID = R.drawable.ic_round_wifi_tethering_24,
        notificationCategory = Notification.CATEGORY_SERVICE,
        isRemovable = false
    )
    override val messages = JNotificationData(
        channelID = "Messages",
        channelName = "Direct messages",
        importance = NotificationManager.IMPORTANCE_MAX,
        notificationVisibility = Notification.VISIBILITY_SECRET,
        smallIconResID = R.drawable.ic_baseline_email_24,
        notificationCategory = Notification.CATEGORY_MESSAGE,
        isRemovable = true
    )
    override val serversScanner = JNotificationData(
        channelID = "Searching",
        channelName = "Searching for a server",
        importance = NotificationManager.IMPORTANCE_MAX,
        notificationVisibility = Notification.VISIBILITY_PUBLIC,
        smallIconResID = R.drawable.ic_round_search_24,
        notificationCategory = Notification.CATEGORY_SERVICE,
        isRemovable = false
    )
    override val connectedToServer = JNotificationData(
        channelID = "Connected",
        channelName = "Connected to the server",
        importance = NotificationManager.IMPORTANCE_MAX,
        notificationVisibility = Notification.VISIBILITY_PUBLIC,
        smallIconResID = R.drawable.app_icon_foreground,

```

```

        notificationCategory = Notification.CATEGORY_SERVICE,
        isRemovable = false
    )
}

package com.edu.alterjuicechat.services

import android.app.Service
import android.content.Intent
import com.alterjuice.utils.RenewableJob
import com.edu.alterjuicechat.domain.NotificationIDs
import
com.edu.alterjuicechat.notifications.notification_creator.ServersScannerNotifica
tionGenerator
import com.edu.alterjuicechat.socket.ServersFinder
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.collect
import org.koin.core.component.KoinComponent
import org.koin.core.component.get

class ServersFinderService : BaseForegroundService(), KoinComponent {
    private val finder = get<ServersFinder>()

    // Jobs to stop/cancel
    private var autoStopJob = RenewableJob(::launchAutoStopJob)
    private var notificationUpdater = RenewableJob(::launchNotificationUpdater)

    // Job Creator
    private fun launchNotificationUpdater() = launch {
        finder.availableServers.collect {
            withContext(Dispatchers.Default){
                sendNotification(it.size)
            }
        }
    }
    // Job Creator
    private fun launchAutoStopJob() = launch {
        delay(60*1_000)
        manuallyStopSelf()
    }
}

```

```

    override fun getNotificationIDRelatedToService(): Int {
        return NotificationIDs.SEARCHING_SERVERS_NOTIFICATION_ID
    }

    private fun sendNotification(countAvailableServers: Int = 0){
        ServersScannerNotificationGenerator.createAndSendNotification(
            context = this,
            countAvailableServers = countAvailableServers
        )
    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int
    {
        val performedStop = onStartExtendedReturnsIfStopPerformed(intent)
        if (performedStop){
            return Service.START_NOT_STICKY
        }
        if (!finder.serverScannerIsWorking.value){
            launch {
                finder.startListening()
            }
        }
        autoStopJob.renew()
        notificationUpdater.renew()
        val notification =
ServersScannerNotificationGenerator.createNotification(this, 0)
        startForeground(getNotificationIDRelatedToService(),
notification.build())
        return Service.START_NOT_STICKY
    }

    override fun manuallyStop() {
        launch { finder.stopListening() }
        notificationUpdater.cancelIt()
        autoStopJob.cancelIt()
        super.manuallyStop()
    }
}

package com.edu.alterjuicechat.services

import android.app.Service

```

```

import android.content.Intent
import com.alterjuice.utils.RenewableJob
import com.edu.alterjuice.socket_server.OnServerStart
import com.edu.alterjuice.socket_server.JServer
import com.edu.alterjuicechat.domain.NotificationIDs
import com.edu.alterjuicechat.domain.ServerConsts
import com.edu.alterjuicechat.domain.model.HostStatus
import com.edu.alterjuicechat.domain.storage.server_data.HostData
import
com.edu.alterjuicechat.notifications.notification_creator.BroadcastingNotificati
onGenerator
import com.edu.alterjuicechat.repo.RuntimeHost
import kotlinx.coroutines.*
import org.koin.core.component.KoinComponent
import org.koin.core.component.get

class BroadcastingService : BaseForegroundService(), KoinComponent {

    private val hostData get() = get<HostData>()

    // Vars to stop/cancel
    private var serverThread: Thread? = null
    private var server: JServer? = null

    override fun getNotificationIDRelatedToService(): Int {
        return NotificationIDs.SERVER_NOTIFICATION_ID
    }
    private var selfPingJob = RenewableJob(::generateSelfPingJob)

    private fun generateSelfPingJob() = launch(Dispatchers.IO) {
        while (true){
            RuntimeHost.notifySelfPinged()
            delay(2_500)
        }
    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int
{
    val stopPerformed = onStartExtendedReturnsIfStopPerformed(intent)
    if (stopPerformed) {
        return Service.START_NOT_STICKY
    }
}

```

```

        startForeground(getNotificationIDRelatedToService(),
BroadcastingNotificationGenerator.createNotificationWithoutText(this).build())
        selfPingJob.renew()
        RuntimeHost.emitNewStatus(HostStatus.Preparing)

serverThread = Thread {
    server = JServer(
        checkAndAddIfUserUIDNotRegisteredYet =
hostData::checkIfUserNotRegisteredYet,
        registerUserUID = hostData::registerIfUserNotRegisteredYet
    ).also {
        try {
            it.start(
                port = ServerConsts.SERVER_PORT,
                serverName = hostData.getServerName(),
                serverUID = hostData.getServerUID(),
                onStarted = onServerStarted
            )
        } catch (e: Exception) {
            handleError(e)
        }
    }
}.also {
    it.start()
}
return Service.START_NOT_STICKY
}

private val onServerStarted = OnServerStart { ip: String, port: Int, name:
String ->
    startForeground(
        getNotificationIDRelatedToService(),

BroadcastingNotificationGenerator.createNotification(this@BroadcastingService,
ip, name).build()
    )
    launch {
        delay(1500)
        RuntimeHost.emitNewStatus(HostStatus.Broadcasting(ip, port))
    }
}

override fun manuallyStop() {

```

```

        RuntimeHost.emitNewStatus(HostStatus.Disabled)
        server?.awaitAndClose()
        serverThread?.interrupt()
        selfPingJob.cancelIt()
        super.manuallyStop()
    }

    private fun handleError(e: Exception){
        e.printStackTrace()
    }
}

package com.edu.alterjuicechat.services

import android.content.Intent
import com.edu.alterjuicechat.domain.NotificationIDs
import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.model.ConnectivityStatus
import com.edu.alterjuicechat.extensions.getSystemNotificationManager
import com.edu.alterjuicechat.extensions.toNotificationMessageWithID
import
com.edu.alterjuicechat.notifications.notification_creator.ConnectedToServerNotif
icationGenerator
import
com.edu.alterjuicechat.notifications.notification_creator.MessageNotificationGen
erator
import com.edu.alterjuicechat.repo.ClientWorker
import com.edu.alterjuicechat.repo.RuntimeConnectivity
import com.edu.alterjuicechat.server_model.dto.entities.MessageDto
import kotlinx.coroutines.*
import org.koin.core.component.KoinComponent
import org.koin.core.component.get

class ConnectedToServerService: BaseForegroundService(), KoinComponent {
    private val clientWorker get() = get<ClientWorker>()

    private var statusCheckerJob: Job? = null
    private var lastStatus: ConnectivityStatus = ConnectivityStatus.NotConnected

    private var isStartedForeground = false

    override fun getNotificationIDRelatedToService(): Int {

```

```

        return NotificationIDs.CONNECTED_TO_SERVER_NOTIFICATION_ID
    }

    private fun connectedToServerServiceNewMessagesHandler(activeChatUID:
    UserID?, newMsg: MessageDto){
        if (activeChatUID != newMsg.from.id) {
            MessageNotificationGenerator.createAndSendNotification(
                this,
                newMsg.toNotificationMessageWithID()
            )
        }
    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int
    {
        val stopPerformed = onStartExtendedReturnsIfStopPerformed(intent)
        if (stopPerformed){
            return START_NOT_STICKY
        }
        clientWorker.addHandler("ConnectedToServerService",
        ::connectedToServerServiceNewMessagesHandler)
        launch {
            clientWorker.tryToConnect()
        }
        statusCheckerJob = launch(Dispatchers.Main) {
            while (true){

                val newStatus = RuntimeConnectivity.statusState.value
                if (lastStatus::class.java == newStatus::class.java &&
                isStartedForeground) {
                    delay(1000)
                    continue
                }
                lastStatus = newStatus
                val notification =
                ConnectedToServerNotificationGenerator.createNotificationWithStatus(this@Connect
                edToServerService, lastStatus).build()
                withContext(Dispatchers.Main){
                    if (!isStartedForeground){
                        startForeground(getNotificationIDRelatedToService(),
                notification)

                        isStartedForeground = true
                    } else {

```

```

getSystemNotificationManager().notify(getNotificationIDRelatedToService(),
notification)
        }
    }
    if (lastStatus is ConnectivityStatus.NotConnected){
        delay(1000)
        manuallyStopSelf()
        break
    }
}
}

return START_STICKY
}

override fun manuallyStop() {
    launch {
        clientWorker.stopAll()
    }
    super.manuallyStop()
}
}

package com.edu.alterjuicechat.services

import android.content.Context
import android.content.Intent
import android.os.IBinder
import androidx.annotation.CallSuper
import androidx.core.app.JobIntentService
import com.edu.alterjuicechat.domain.Consts
import
com.edu.alterjuicechat.notifications.notification_creator.cancelNotification
import kotlinx.coroutines.*

/** MyServiceWithNotification is a base class-wrapper for
 * android.Service class.
 * it contains standard functionality to work with coroutines,
 * provides methods to stop services manually so you won't be
 * nervous about the leaks and working jobs you cannot to cancel.
 *
 *
 */

```

```

* Start the service as usual.
* If you want to stop the service - you should call #getIntentToStopService()
* to get the intent with the action which will be handled within the service
* (in #onStartExtendedReturnsIfStopPerformed() function)
*
* So all you need is to implement #getNotificationIDRelatedToService() if it is
needed
* and
* call #onStartExtendedReturnsIfStopPerformed() on the start of the
#onStartCommand() function.
* The result which #onStartExtendedReturnsIfStopPerformed() returns -> is the
signal whether the
* stop was performed, so user can not continue the #onStartCommand().
* */
abstract class BaseForegroundService: JobIntentService() {
    private val job = SupervisorJob()
    private val scope = CoroutineScope(Dispatchers.Default + job)
    private var manuallyStopped = false

    /** Default launch function written to encapsulate the "scope" variable.
    * */
    protected fun BaseForegroundService.launch(
        dispatchers: CoroutineDispatcher = Dispatchers.Default,
        block: suspend CoroutineScope.() -> Unit
    ): Job = scope.launch(dispatchers) { block() }

    /** Returns the NotificationID to destroy notification on #manuallyStop().
    * Useful for services with static notificationID e.g Scanner or
Broadcasting. */
    protected open fun getNotificationIDRelatedToService(): Int? = null

    /** ManuallyStop function is used to stop all the declared jobs and clear
the state.
    * after that the service might be destroyed.
    * Super call is required to stop the jobs and scopes
    * */
    @CallSuper
    open fun manuallyStop(){
        getNotificationIDRelatedToService()?.let{ cancelNotification(this, it) }
        runBlocking {

```

```

        job.cancel()
        scope.cancel()
    }
    manuallyStopped = true
}

/** Overloaded onDestroy to manipulate clearing of state -> not to call
#manuallyStop() twice */
override fun onDestroy() {
    if (!manuallyStopped){
        manuallyStop()
    }
    super.onDestroy()
}

override fun onHandleWork(intent: Intent) { /* Do nothing */ }
override fun onBind(intent: Intent): IBinder? { return null /* Do nothing */
}

/** An extended functionality for #onStartCommand() call
 * to check if Intent#action equals with the action to stop the service.
 * If it is an intent to stop the server -> manuallyStop is called
 * and returns true (like "yep, stop was performed"), otherwise false
 * and you can continue to handle onStartCommand
 * */
protected fun onStartExtendedReturnsIfStopPerformed(intent: Intent?):
Boolean {
    if (intent?.action == getStopMyServiceAction(this::class.java)){
        manuallyStopSelf()
        return true
    }
    return false
}
protected fun manuallyStopSelf(){
    manuallyStop()
    stopSelf()
}
}

inline fun <reified T: BaseForegroundService> getIntentToStopService(context:
Context): Intent {

```

```

        return Intent(context, T::class.java).also {
            it.action = getStopMyServiceAction(T::class.java)
        }
    }
}
fun getStopMyServiceAction(clazz: Class<out BaseForegroundService>): String {
    return "${Consts.APP_PACKAGE}.STOP_SERVICE_${clazz.simpleName}"
}

```

```
package com.edu.alterjuicechat.domain.storage.server_data
```

```
import com.edu.alterjuicechat.domain.ServerName
import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.domain.UserUID
```

```
data class HostDataState(
    val serverUID: ServerUID? = null,
    val serverName: ServerName? = null,
    val allRegisteredUsers: List<UserUID> = emptyList()
)

```

```
data class HostDataStateNonNull(
    val serverUID: ServerUID,
    val serverName: ServerName
)

```

```
package com.edu.alterjuicechat.domain.storage.server_data
```

```
import com.edu.alterjuicechat.domain.ServerName
import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.domain.UserUID
import com.edu.alterjuicechat.domain.storage.StorageStateFlow
```

```
interface HostData: StorageStateFlow<HostDataState>{
    fun revokeServerUID(): ServerUID

    fun getServerUID(): ServerUID
    fun getServerName(): ServerName

    fun setServerUID(serverUID: ServerUID)
    fun setServerName(serverName: ServerName)
    fun checkIfUserNotRegisteredYet(userUID: UserUID): Boolean
    fun registerIfUserNotRegisteredYet(userUID: UserUID)
}

```

```
}
```

```
package com.edu.alterjuicechat.domain.storage.session_data.dynamic
```

```
import com.edu.alterjuicechat.domain.ServerUID
```

```
import com.edu.alterjuicechat.domain.storage.StorageStateFlow
```

```
interface SessionDynamicData: StorageStateFlow<SessionDynamicDataState>{
```

```
    fun getSessionUID(): ServerUID?
```

```
    fun setSessionUID(newSessionUID: ServerUID)
```

```
    fun updateWithState(state: SessionDynamicDataState)
```

```
    fun transformState(block: (SessionDynamicDataState) ->
SessionDynamicDataState)
```

```
}
```

```
package com.edu.alterjuicechat.domain.storage.session_data.dynamic
```

```
import com.alterjuice.utils.getCurrentDate
```

```
import com.edu.alterjuicechat.domain.ServerUID
```

```
import com.edu.alterjuicechat.domain.UserDisplayName
```

```
import com.edu.alterjuicechat.domain.UserUID
```

```
import com.edu.alterjuicechat.domain.model.*
```

```
import com.edu.alterjuicechat.server_model.dto.entities.MessageDto
```

```
import com.edu.alterjuicechat.server_model.dto.entities.UserDto
```

```
fun MessageDto.toSimpleMessage(): SimpleMessage {
```

```
    return SimpleMessage(this.from.id, this.message, getCurrentDate())
```

```
}
```

```
fun SimpleMessage.toMessage(userName: UserDisplayName): Message {
```

```
    return Message(this.fromUID, userName, message, date)
```

```
}
```

```
data class SessionDynamicDataState(
```

```
    val sessionUID: ServerUID? = null,
```

```
    val data: MutableMap<UserUID, ChatInfo> = mutableMapOf(),
```

```
) {
```

```
    private val EMPTY_CHAT = DataChat("", "")
```

```

    private val onlineChatsMap: MutableMap<UserUID, Boolean> =
mutableMapOf<UserUID, Boolean>().apply {
    putAll(data.entries.map { it.key to false })
}

fun getChatIsOnline(userUID: UserUID): Boolean {
    return onlineChatsMap[userUID] == true
}

fun createUserIfNeeded(userUID: UserUID, userName: UserDisplayName){
    if (data.containsKey(userUID))
        return
    data[userUID] = ChatInfo(userUID, userName)
}

fun getChats(): List<Chat> = data.keys.map(::getChat)

fun getOnlineChats(): List<Chat> =
data.keys.filter(::getChatIsOnline).map(::getChat)

fun addMessage(userUID: UserUID, message: MessageDto, increaseCounter:
Boolean) {
    val chat = data[userUID]?: ChatInfo(userUID, "")
    val copy = chat.copy(messages = chat.messages +
message.toSimpleMessage()).let {
        if (increaseCounter)
            it.copy(unreadMessagesCount = it.unreadMessagesCount + 1)
        else it
    }
    data[userUID] = copy
}

fun getChat(userUID: UserUID): Chat {
    val chatInfo = data[userUID]
    if (chatInfo == null)
        return EMPTY_CHAT
    val lastMessage = getLastMessage(userUID)
    val isOnline = onlineChatsMap[userUID] ?: false

    if (lastMessage == null)
        return DataChat(chatInfo.userUID, chatInfo.userName, isOnline =
isOnline)
}

```

```

        return DataChat(chatInfo.userID, chatInfo.userName, lastMessage,
chatInfo.unreadMessagesCount, isOnline)
    }

    fun clearUnreadCounter(userID: UserID){
        val chat = data[userID]
        chat?.let {
            data[userID] = it.copy(unreadMessagesCount = 0)
        }
    }

    fun handleNewUsers(newUsers: List<UserDto>) {
        val newUsersIDs = newUsers.map(UserDto::id).toSet()
        val existedUsersIDs = data.keys.toSet()
        val notAddedUsers = newUsersIDs.minus(existedUsersIDs)
        val notActiveAnymoreUsers = existedUsersIDs.minus(newUsersIDs)

        notAddedUsers.forEach { uid ->
            createUserIfNeeded(uid, newUsers.find { it.id == uid }!!.name)
            onlineChatsMap[uid] = true
        }
        newUsersIDs.forEach { uid ->
            onlineChatsMap[uid] = true
        }
        notActiveAnymoreUsers.forEach { uid ->
            onlineChatsMap[uid] = false
        }
    }

    fun getMessagesCount(userID: UserID): Int {
        return data[userID]?.messages?.size?: 0
    }

    fun getMessages(userID: UserID): List<Message> {
        val chat = data[userID]?: return emptyList()
        val messages = chat.messages
        return messages.map { it.toMessage(chat.userName) }
    }

    fun getLastMessage(userID: UserID): LastMessage? {
        val chat = data[userID]?: return null
        if (chat.messages.isEmpty()) return null
    }

```

```

        val lastMessage = chat.messages.last().toMessage(chat.userName)
        return LastMessage(lastMessage.text, lastMessage.date)
    }

    fun increaseUnreadMessagesCountBy(userID: UserID, increaseBy: Int) {
        val chat = data[userID]
        data[userID]?.unreadMessagesCount = (chat?.unreadMessagesCount?: 0) +
increaseBy
    }
}

package com.edu.alterjuicechat.domain.storage.session_data.static

import com.edu.alterjuicechat.domain.ServerUID

data class SessionStaticDataState(
    val sessionUID: ServerUID? = null,
)

package com.edu.alterjuicechat.domain.storage.session_data.static

import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.domain.storage.StorageStateFlow
import
com.edu.alterjuicechat.domain.storage.session_data.dynamic.SessionDynamicData

interface SessionStaticData: StorageStateFlow<SessionStaticDataState>{
    fun getSessionUID(): ServerUID
    fun setSessionUID(newSessionUID: ServerUID?)
}

package com.edu.alterjuicechat.domain.storage

import kotlinx.coroutines.flow.StateFlow

interface StorageStateFlow<TState> {
    val state: StateFlow<TState>
}

package com.edu.alterjuicechat.domain.storage.user_data

import com.edu.alterjuicechat.domain.storage.StorageStateFlow

```

```
interface UserData: StorageStateFlow<UserDataState>{
    fun getDisplayName(): String

    fun setDisplayName(displayName: String)
}
```

```
package com.edu.alterjuicechat.domain.storage.user_data
```

```
data class UserDataState(
    val displayName: String? = null
)
```

```
package com.edu.alterjuicechat.domain.model
```

```
import com.edu.alterjuicechat.domain.ServerUID
import com.edu.alterjuicechat.server_model.dto.ServerParams
```

```
sealed interface ConnectivityStatus {
    object NotConnected: ConnectivityStatus
```

```
    class ChosenServerToConnect(
        override val ip: String,
        override val name: String,
        override val serverUID: ServerUID
    ) : ConnectivityStatus, ServerParams {
        constructor(params: ServerParams): this(params.ip, params.name,
params.serverUID)

        fun toConnectedToServer() = Connected.ConnectedToServer(this)
    }

    sealed class Connected(
        override val ip: String,
        override val name: String,
        override val serverUID: ServerUID
    ) : ConnectivityStatus, ServerParams {
        constructor(params: ServerParams): this(params.ip, params.name,
params.serverUID)
```

```

        class ConnectedToServer(params: ServerParams): Connected(params),
ServerParams {
            fun toAuthenticated(finalSessionID: String) = Authenticated(this,
finalSessionID)
        }

        class Authenticated(params: ServerParams, val finalSessionID: String):
Connected(params), ServerParams
        }

    }

```

```
package com.edu.alterjuicechat.domain.model
```

```
data class LastMessage(val message: String?, val dateInMs: Long)
```

```
package com.edu.alterjuicechat.domain.model
```

```
import com.edu.alterjuicechat.domain.UserUID
```

```

interface Chat: User {
    var lastMessage: LastMessage
    var unreadMessagesCount: Int
    var isOnline: Boolean
}

```

```

data class DataChat constructor(
    override val userID: UserUID,
    override val userName: String,
    override var lastMessage: LastMessage = LastMessage("", 0),
    override var unreadMessagesCount: Int = 0,
    override var isOnline: Boolean = false
): Chat {
    override fun toString(): String {
        return "Name: $userName; ID: $userID; lastMessage: $lastMessage"
    }
}

```

```
package com.edu.alterjuicechat.domain.model
```

```

import com.edu.alterjuicechat.domain.UserDisplayName
import com.edu.alterjuicechat.domain.UserUID

```

```

data class ChatInfo(
    val userID: UserID,
    val userName: UserDisplayName,
    val messages: List<SimpleMessage> = mutableListOf(),
    var unreadMessagesCount: Int = 0,
)

package com.edu.alterjuicechat.domain.model

import com.edu.alterjuicechat.domain.UserID

data class SimpleMessage(val fromUID: UserID, val message: String, val date:
Long)

package com.edu.alterjuicechat.domain.model

import com.alterjuice.utils.isOneOf

sealed interface HostStatus {
    object Disabled: HostStatus
    object Preparing: HostStatus
    data class ClosedWithError(val error: String? = null): HostStatus
    data class Broadcasting(val ip: String, val port: Int): HostStatus
    fun isPreparingOnBroadcasting(): Boolean {
        return this isOneOf listOf(Preparing::class.java,
Broadcasting::class.java)
    }
}

package com.edu.alterjuicechat.domain.model

import com.edu.alterjuicechat.domain.UserID

interface User {
    val userID: UserID
    val userName: String
}

data class DataUser constructor(
    override val userID: String,
    override val userName: String,
): User

```

```

package com.edu.alterjuicechat.domain.model

import com.edu.alterjuicechat.server_model.dto.ServerParamsWithSize

data class ServerInfo constructor(
    override val ip: String,
    override val name: String,
    override val serverUID: String,
    override val clientsSize: Int = 0
): ServerParamsWithSize {
    constructor(params: ServerParamsWithSize):
        this(params.ip, params.name, params.serverUID, params.clientsSize)
}

package com.edu.alterjuicechat.domain.model

data class Message constructor(
    val senderId: String,
    val senderName: String,
    val text: String,
    val date: Long = 0L
)

package com.edu.alterjuicechat.domain

object Consts {
    const val APP_PACKAGE = "com.edu.alterjuicechat"
    const val APP_GITHUB_CREATOR_LINK = "https://github.com/alterjuice"
    const val APP_GITHUB_CREATOR_SHORT_INFO = "Github/AlterJuice"

    // Why emulators are not receiving packets over udp: use another ip
    // https://stackoverflow.com/a/41981731
    const val SIMPLE_MESSAGE_DATE_FORMAT = "k:mm"
}

object NotificationIDs {
    const val SERVER_NOTIFICATION_ID = 141246
    const val SEARCHING_SERVERS_NOTIFICATION_ID = 195136
    const val CONNECTED_TO_SERVER_NOTIFICATION_ID = 159319
    var MESSAGE_NOTIFICATION_ID = 0
}

```

```

}

object NavArgKeys {
    const val FILTER_CHATS_BY = "filterBy"
    const val CHAT_USER_ID = "userID"
    const val CHAT_USER_NAME = "userName"
}

object ServerConsts {
    const val SERVER_UDP_KEY_TO_GET_TCP_IP = "Send ip'des"
    const val SERVER_PORT = 6666
    const val UDP_DELAY = 0L
    const val UDP_PORT = 8888
    const val UDP_TIMEOUT = 5000
    const val UDP_PACKET_SIZE = 256
    private const val UDP_ADDRESS_FOR_EMULATOR = "10.0.2.2"
    private const val UDP_ADDRESS_FOR_DEVICES = "255.255.255.255"

    val UDP_ADDRESS_BOTH = listOf(
        UDP_ADDRESS_FOR_EMULATOR,
        UDP_ADDRESS_FOR_DEVICES
    )
}

package com.edu.alterjuicechat.domain

typealias ServerUID = String
typealias ServerName = String
typealias ServerIP = String
typealias UserDisplayName = String
typealias UserID = String

package com.edu.alterjuicechat.server_model.dto

import com.edu.alterjuicechat.server_model.dto.entities.*
import com.google.gson.Gson

class GeneratorDto(private val gson: Gson) {

    /* ID is SessionID got from first TCP socket connection. */

```

```

    fun generatePing(id: String): BaseDto {
        return generateBaseDto(BaseDto.Action.PING, PingDto(id))
    }

    fun generateSendMessage(id: String, receiver: String, message: String):
BaseDto {
        return generateBaseDto(BaseDto.Action.SEND_MESSAGE, SendMessageDto(id,
receiver, message))
    }

    fun generateConnect(id: String, connectName: String): BaseDto {
        return generateBaseDto(BaseDto.Action.CONNECT, ConnectDto(id,
connectName))
    }

    fun generateDisconnect(id: String, code: Int): BaseDto {
        return generateBaseDto(BaseDto.Action.DISCONNECT, DisconnectDto(id,
code))
    }

    fun generateGetUsers(id: String): BaseDto {
        return generateBaseDto(BaseDto.Action.GET_USERS, GetUsersDto(id))
    }

    fun generateAuthAs(initialSessionID: String, oldSessionID: String?): BaseDto
{
        return generateBaseDto(
            BaseDto.Action.AUTH_AS,
            AuthAsDto(initialSessionID, authWithUID = oldSessionID)
        )
    }

    fun toJson(obj: Payload): String {
        return gson.toJson(obj)
    }

    fun toJson(obj: BaseDto): String {
        return gson.toJson(obj)
    }

    fun generateBaseDto(action: BaseDto.Action, payload: Payload): BaseDto {
        return BaseDto(action, toJson(payload))
    }
}

package com.edu.alterjuicechat.server_model.dto

```

```

interface ServerParamsWithSize: ServerParams {
    val clientsSize: Int
}

interface ServerParams {
    val ip: String
    val name: String
    val serverUID: String
}

package com.edu.alterjuicechat.server_model.dto

import com.edu.alterjuicechat.server_model.dto.entities.*
import com.google.gson.Gson

class ParserDto(private val gson: Gson){
    fun parseMessage(baseDtoMessagePayload: String): MessageDto {
        return convertPayloadTo<MessageDto>(baseDtoMessagePayload)
    }
    fun parseUdp(payload: String): UdpDto {
        return convertPayloadTo<UdpDto>(payload)
    }
    fun parseUsersReceived(payload: String): UsersReceivedDto {
        return convertPayloadTo<UsersReceivedDto>(payload)
    }
    fun parseConnected(baseDtoConnectedPayload: String): ConnectedDto {
        return convertPayloadTo<ConnectedDto>(baseDtoConnectedPayload)
    }
    fun parseBaseDto(responsePayload: String): BaseDto {
        return convertPayloadTo<BaseDto>(responsePayload)
    }
    fun parsePong(payload: String): PongDto {
        return convertPayloadTo<PongDto>(payload)
    }

    private inline fun <reified T> convertPayloadTo(payload: String): T {
        return gson.fromJson(payload, T::class.java)
    }
}

package com.edu.alterjuicechat.server_model.dto.entities

```

```

/** User can auth with its own ID if he has from the last session.
 * Otherwise it can be null and server will return the new UID
 *
 * */
data class AuthAsDto(val initUID: String, val authWithUID: String? = null) :
    Payload

package com.edu.alterjuicechat.server_model.dto.entities

import com.google.gson.Gson

interface Payload

fun Payload.toJson(gson: Gson): String{
    return gson.toJson(this)
}

package com.edu.alterjuicechat.server_model.dto.entities

data class PingDto(val id: String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class MessageDto(val from: UserDto, val message: String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class ConnectedDto(val id: String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class UserDto(val id: String, val name: String): Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class Error(val message: String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class UsersReceivedDto(val users: List<UserDto>) : Payload

```

```

package com.edu.alterjuicechat.server_model.dto.entities

data class PongDto(val id: String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class DisconnectDto(val id: String, val code: Int) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class SendMessageDto(val id: String, val receiver: String, val message:
String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class BaseDto(val action: Action, val payload: String) {

    enum class Action {
        PING, PONG, AUTH_AS, CONNECT, CONNECTED, GET_USERS, USERS_RECEIVED,
        SEND_MESSAGE, NEW_MESSAGE, DISCONNECT
    }
}

package com.edu.alterjuicechat.server_model.dto.entities

data class GetUsersDto(val id: String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

data class ConnectDto(val id: String, val name: String) : Payload

package com.edu.alterjuicechat.server_model.dto.entities

import com.edu.alterjuicechat.server_model.dto.ServerParamsWithSize

data class UdpDto constructor(
    override val ip: String,
    override val name: String,
    override val serverUID: String,
    override val clientsSize: Int = 0
) : Payload, ServerParamsWithSize

```

```

package com.edu.alterjuicechat.server_model

import com.edu.alterjuicechat.server_model.dto.GeneratorDto
import com.edu.alterjuicechat.server_model.dto.ParserDto
import com.google.gson.Gson
import org.koin.dsl.module

object DI {
    val modules = module {
        single { Gson() }
        single { ParserDto(get<Gson>()) }
        single { GeneratorDto(get<Gson>()) }
    }
}

package com.edu.alterjuicechat.socket

import com.edu.alterjuicechat.server_model.dto.ServerParamsWithSize
import kotlinx.coroutines.flow.StateFlow

interface ServersFinderState {
    val availableServers: StateFlow<List<ServerParamsWithSize>>
    val serverScannerIsWorking: StateFlow<Boolean>
}

package com.edu.alterjuicechat.socket

import com.alterjuice.utils.RenewableJob
import com.edu.alterjuicechat.domain.ServerConsts
import com.edu.alterjuicechat.domain.ServerIP
import com.edu.alterjuicechat.domain.model.ServerInfo
import com.edu.alterjuicechat.server_model.dto.ParserDto
import com.edu.alterjuicechat.server_model.dto.ServerParamsWithSize
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.*
import java.io.IOException
import java.net.DatagramPacket
import java.net.DatagramSocket
import java.net.InetAddress

```

```

import java.net.SocketTimeoutException

interface ServersFinder: ServersFinderManager, ServersFinderState

internal class ServersFinderImpl(private val parser: ParserDto): ServersFinder {
    private val servers: HashMap<ServerIP, ServerInfo> = HashMap()
    override val serverScannerIsWorking = MutableStateFlow(false)
    override val availableServers =
MutableStateFlow<List<ServerParamsWithSize>>(emptyList())
    private val job = SupervisorJob()
    private val scope = CoroutineScope(Dispatchers.Default + job)
    private var scannerJob = RenewableJob(::listenToServers)

    override suspend fun startListening(){
        serverScannerIsWorking.emit(true)
        availableServers.emit(emptyList())
        scannerJob.renew()
    }

    override suspend fun stopListening(){
        serverScannerIsWorking.emit(false)
        servers.clear()
        scannerJob.cancelIt()
    }

    private fun listenToServers(
        udpPort: Int = ServerConsts.UDP_PORT,
    ): Job {
        return scope.launch(Dispatchers.IO) {
            var attempts = 0
            val messageToSend =
ServerConsts.SERVER_UDP_KEY_TO_GET_TCP_IP.encodeToByteArray()
            val datagramSocket = DatagramSocket().apply { soTimeout =
ServerConsts.UDP_TIMEOUT }
            val receiveBuffer = ByteArray(ServerConsts.UDP_PACKET_SIZE)
            val receivePacket = DatagramPacket(receiveBuffer,
receiveBuffer.size)
            var udpAddress: String
            while (serverScannerIsWorking.value) {
                delay(1000)
                attempts += 1

```



```

    }
}

package com.edu.alterjuicechat.socket

interface ServersFinderManager {
    suspend fun startListening()
    suspend fun stopListening()
}

package com.edu.alterjuice.socket_server

import com.edu.alterjuicechat.server_model.dto.entities.UserDto as User
import com.edu.alterjuicechat.server_model.dto.entities.*
import com.google.gson.Gson
import kotlinx.coroutines.*
import com.edu.alterjuicechat.server_model.dto.entities.BaseDto.Action.*
import com.edu.alterjuicechat.server_model.dto.entities.ConnectedDto
import com.edu.alterjuicechat.server_model.dto.entities.PingDto
import com.edu.alterjuicechat.server_model.dto.entities.UdpDto
import org.koin.core.component.KoinComponent
import java.net.*
import java.util.*

fun interface OnServerStart {
    fun invoke(ip: String, port: Int, name: String)
}

class JServer constructor(
    private val checkAndAddIfUserUIDNotRegisteredYet: (old: String) -> Boolean,
    private val registerUserUID: (userID: String) -> Unit
): KoinComponent {
    var isRunning = false
    private var server: ServerSocket? = null // lateinit
    private val job = SupervisorJob()
    private val coroutineScope = CoroutineScope(Dispatchers.Default + job)
    private val clients = mutableListOf<JClientHolder>()
    private val gson = Gson()

    fun start(
        port: Int,
        serverName: String,

```

```

        serverUID: String = UUID.randomUUID().toString(),
        onStart: OnServerStart
    ) {
        startServer(port, serverName, serverUID){ ip, port, name ->
            println("##### HOST<${serverName}> MSG: Created $ip $port $name")
            onStart.invoke(ip, port, name)
        }
    }
}

private fun startServer(port: Int, serverName: String, serverUID: String,
onStart: OnServerStart) {
    isRunning = true
    coroutineScope.launch(Dispatchers.IO) {
        server = ServerSocket(port).also {
            it.reuseAddress = true
        }
        val s = DatagramSocket()
        s.connect(InetAddress.getByName("8.8.8.8"), 10002)
        val address = s.localAddress
        val serverIP = address.hostAddress
        onStart.invoke(serverIP, port, serverName)
        startUdp(serverIP, serverName, serverUID, clients::size)
        println("##### HOST<${serverName}> MSG: server started at port
$serverIP:$port")
        while (true) {
            var clientAttempts = 0
            // val client = server.accept()
            val client = try {
                server?.accept() ?: continue
            } catch (socketException: SocketException){
                clientAttempts += 1
                if (clientAttempts >= 5){
                    awaitAndClose()
                    break
                }
            }
            continue
        } catch (e: java.lang.Exception){
            e.printStackTrace()
            continue
        }
        val clientHandler = JSocketHandler(client,
createClientListener(), coroutineScope)

```

```

        println("##### HOST<${serverName}> MSG: ____connected:
${client.inetAddress}")
        clientHandler.loop()
        val initUUID = UUID.randomUUID().toString()
        val JClientHolder = JClientHolder(clientHandler, User(initUUID,
""), isAuthenticated = false)
        JClientHolder.startConnectCountdown(coroutineScope, 10000) {
holder ->
            holder.jSocketHandler.disconnect()
            clients.removeIf { it.user.id == holder.user.id }
        }
        delay(1000)
        clientHandler.send(gson.toJson(BaseDto(CONNECTED,
gson.toJson(ConnectedDto(initUUID))))))
        clients.add(JClientHolder)
        JClientHolder.restartPingTimeout(coroutineScope, onTimeout = {
client ->
            client.jSocketHandler.disconnect()
            clients.remove(client)
        })
        println("##### HOST<${serverName}> MSG: client added")
    }
}
}
private var udpServer: DatagramSocket? = null

private fun startUdp(serverIP: String, serverName: String, serverUID:
String, getClientsSize: () -> Int) {
    coroutineScope.launch {
        while (true) {
            var port = 8888
            while (true){
                println("TRYING_PORT_${port}")
                udpServer = try {
                    DatagramSocket(port).also {
                        it.reuseAddress = true
                    }
                } catch (e: Exception){
                    println("udp tryingPort_exception ${e.message}")
                    port += 1
                    continue
                }
            }
            break
        }
    }
}

```

```

    }
    println("##### HOST<${serverName}> MSG: udp started at
${udpServer?.port}")
    while (true) {
        try {
            val buffer = ByteArray(256)
            var packet = DatagramPacket(buffer, buffer.size)
            udpServer?.receive(packet)
            val received = packet.data.decodeToString()
            println("##### HOST MSG: udp received:
${received.slice(0..10)}")
            val response = gson.toJson(UdpDto(serverIP, serverName,
serverUID, getClientsSize()))
                .toArray()
            udpServer?.send(
                DatagramPacket(
                    response,
                    response.size,
                    packet.address,
                    packet.port
                )
            )
        } catch (e: SocketException){
            println("##### HOST<${serverName}> MSG: error
handling udp SocketException: $e")
            udpServer?.close()
            break

        } catch (e: Exception) {
            // udpServer.close()
            println("##### HOST<${serverName}> MSG: error
handling udp: $e")
            udpServer?.close()
            break
        }
    }
}
}
}

private fun createClientListener(): SocketListener = object : SocketListener
{
    override fun onNewMessage(message: String) {

```

```

    try {
        val dto = convertPayloadTo<BaseDto>(message)
        when (dto.action) {
            AUTH_AS -> {
                handleAuth(dto.payload)
            }
            PING -> {
                handlePing(dto.payload)
            }
            CONNECT -> {
                handleConnect(dto.payload)
            }
            SEND_MESSAGE -> {
                sendMessage(dto.payload)
            }
            GET_USERS -> {
                handleGetUsers(dto.payload)
            }
            DISCONNECT -> {
                handleDisconnect(dto.payload)
            }
            else -> println("unknown action: $message")
        }
    } catch (e: Exception) {
        println("error handling message: ${e.message}")
    }
}

private inline fun <reified T> convertPayloadTo(payload: String): T {
    return gson.fromJson(payload, T::class.java)
}

private fun handleAuth(payload: String) {
    val authData = convertPayloadTo<AuthAsDto>(payload)
    val authWithUID = authData.authWithUID

    val finalClientUID: String
    if (authWithUID.isNullOrEmpty()){
        finalClientUID = authData.initUID
    } else {
        val isAlreadyRegistered =
checkAndAddIfUserUIDNotRegisteredYet(authWithUID)
    }
}

```

```

        if (isAlreadyRegistered){
            finalClientUID = authWithUID
        } else {
            finalClientUID = authData.initUID
        }
    }
    clients.firstOrNull { it.user.id == authData.initUID }?.also {
        it.user = it.user.copy(id = finalClientUID)
        it.isAuthenticated = true
        it.jSocketHandler.send(gson.toJson(BaseDto(CONNECTED,
gson.toJson(ConnectedDto(finalClientUID))))))
    }
}

private fun handlePing(payload: String) {
    val pingData = convertPayloadTo<PingDto>(payload)
    clients.firstOrNull { it.user.id == pingData.id }?.also {
        it.restartPingTimeout(coroutineScope, onTimeout = { client ->
            client.jSocketHandler.disconnect()
            clients.remove(client)
        })
        it.jSocketHandler.send(PongDto(pingData.id).toJson(PONG))
    }
}

private fun handleConnect(payload: String) {
    val connectData = convertPayloadTo<ConnectDto>(payload)
    clients.firstOrNull { it.user.id == connectData.id }?.also {
        registerUserUID(connectData.id)
        it.stopConnectCountdown()
        it.user = User(connectData.id, connectData.name)
        it.isAuthenticated = true
        it.restartPingTimeout(coroutineScope, onTimeout = { client ->
            client.jSocketHandler.disconnect()
            clients.remove(client)
        })
    }
}

private fun sendMessage(payload: String) {
    val sendData = convertPayloadTo<SendMessageDto>(payload)
    clients.firstOrNull { it.user.id == sendData.receiver }?.also { receiver
->

```

```

        clients.firstOrNull { it.user.id == sendData.id }?.also { sender ->
            val message = MessageDto(sender.user, sendData.message)
            receiver.jSocketHandler.send(gson.toJson(BaseDto(NEW_MESSAGE,
gson.toJson(message))))
        }
    }
}

private fun handleGetUsers(payload: String) {
    val getUsersDto = convertPayloadTo<GetUsersDto>(payload)
    val users = mutableListOf<User>()
    var requester: JClientHolder? = null
    clients.filter { it.isAuthenticated }.forEach {
        if (it.user.id == getUsersDto.id) {
            requester = it
        } else {
            users.add(it.user)
        }
    }
    val usersReceived = UsersReceivedDto(users)
    requester?.also {
        it.jSocketHandler.send(gson.toJson(BaseDto(USERS_RECEIVED,
gson.toJson(usersReceived))))
    }
}

fun handleDisconnect(payload: String) {
    val data = convertPayloadTo<DisconnectDto>(payload)
    val client = clients.firstOrNull { it.user.id == data.id }
    if (client != null) {
        client.jSocketHandler.disconnect()
        println("user ${client.user.id} disconnected")
        clients.remove(client)
    }
}

fun clearClients() {
    clients.forEach {
        it.jSocketHandler.disconnect()
    }
    clients.clear()
}

```

```

        fun awaitAndClose() {
            clearClients()
            runBlocking {
                job.cancel()
                udpServer?.close()
            }
            server?.close()
            isRunning = false
        }
    }

}

package com.edu.alterjuice.socket_server

import com.edu.alterjuicechat.server_model.dto.entities.BaseDto
import com.google.gson.Gson
import com.edu.alterjuicechat.server_model.dto.entities.Payload

private val gson = Gson()

fun Payload.toJson(action: BaseDto.Action): String = gson.toJson(
    BaseDto(action = action, payload = gson.toJson(this))
)

package com.edu.alterjuice.socket_server

import kotlinx.coroutines.*
import java.net.InetSocketAddress
import java.net.Socket
import java.net.SocketException

class JClient(onNewPayload: (msg: String) -> Unit) {

    private var socket: Socket? = null
    private var isRunning = true
    private var eventsHandler: (msg: String) -> Unit = onNewPayload
    private var jSocketHandler: JSocketHandler? = null

    private val defaultListener = object : SocketListener {
        override fun onNewMessage(message: String) {
            eventsHandler.invoke(message)
        }
    }

```

```

    }

    fun initialize(scope: CoroutineScope, ip: String, port: Int){
        try {
            recreateSockets(scope, ip, port)
        } catch (e: SocketException){
            recreateSockets(scope, ip, port)
        }
    }

    private fun recreateSockets(scope: CoroutineScope, ip: String, port: Int){
        socket?.close()
        jSocketHandler?.disconnect()
        socket = Socket().also {
            it.connect(InetSocketAddress(ip, port))
            jSocketHandler = JSocketHandler(it, defaultListener, scope).also {
handler ->
                handler.loop()
            }
        }
    }

    fun send(message: String) {
        jSocketHandler?.send(message)
    }

    fun awaitAndClose() {
        isRunning = false
        jSocketHandler?.disconnect()
        socket?.close()
    }

    fun setNewMessagesHandler(block: (msg: String) -> Unit) {
        eventsHandler = block
    }

}

package com.edu.alterjuice.socket_server

interface SocketListener {
    fun onNewMessage(message: String)
}

```

```

package com.edu.alterjuice.socket_server

import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.launch
import java.io.*
import java.net.Socket

const val CLIENT_TIMEOUT = 20000

class JSocketHandler(
    private val client: Socket,
    private val listener: SocketListener,
    private val scope: CoroutineScope
) {

    private lateinit var reader: BufferedReader
    private lateinit var writer: PrintWriter
    private lateinit var input: InputStream
    private lateinit var clientLoopJob: Job
    private var connected = true

    fun send(message: String) {
        scope.launch(Dispatchers.IO) {
            writer.println(message)
            writer.flush()
        }
    }

    fun loop() {
        client.soTimeout = CLIENT_TIMEOUT
        clientLoopJob = scope.launch(Dispatchers.IO) {
            input = client.getInputStream()
            reader = BufferedReader(InputStreamReader(client.getInputStream()))
            writer = PrintWriter(OutputStreamWriter(client.getOutputStream()))
            while (connected) {
                read()
            }
        }
    }
}

```

```

private fun read() {
    try {
        val data = reader.readLine() ?: return
        listener.onNewMessage(data)
    } catch (io: IOException) {
        disconnect()
    } catch (e: Exception) {
        e.printStackTrace()
    }
}

fun disconnect() {
    connected = false
    clientLoopJob.cancel()
    input.close()
    writer.close()
    client.close()
}

}

package com.edu.alterjuice.socket_server

import com.edu.alterjuicechat.server_model.dto.entities.UserDto as User
import kotlinx.coroutines.*

data class JClientHolder(val jSocketHandler: JSocketHandler, var user: User, var
isAuthenticated: Boolean) {

    private var connectionTimeoutJob: Job? = null
    private var pingTimeoutJob: Job? = null

    fun startConnectCountdown(scope: CoroutineScope, timeout: Long, onTimeout:
(JClientHolder) -> Unit) {
        println("start connection countdown")
        connectionTimeoutJob = scope.launch(Dispatchers.IO) {
            delay(timeout)
            onTimeout(this@JClientHolder)
        }
    }

    fun restartPingTimeout(scope: CoroutineScope, timeout: Long = 15_000,
onTimeout: (JClientHolder) -> Unit) {
        pingTimeoutJob?.cancel()
    }
}

```

```

        pingTimeoutJob = scope.launch(Dispatchers.IO) {
            delay(timeout)
            onTimeout(this@JClientHolder)
        }
    }

    fun stopConnectCountdown() {
        println("stop connection countdown")
        connectionTimeoutJob?.cancel()
    }
}

package com.alterjuice.utils

fun Int.half(): Float = this / 2f

inline fun String?.ifNullOrEmpty(block: () -> String): String {
    if (this.isNullOrEmpty())
        return block()
    return this
}

fun <T> List<T>.contains(predicate: (T) -> Boolean): Boolean {
    this.forEach { if (predicate(it)){ return true } }
    return false
}

package com.alterjuice.utils

import java.lang.Exception
import java.util.*

fun generateUUID(): String {
    return UUID.randomUUID().toString()
}

fun getCurrentDate(): Long {
    return Date().time
}

```

```

fun whileIndexed(condition: (index: Int) -> Boolean): ((Int) -> Unit) -> Unit {
    return { block ->
        var i = 0
        while (condition.invoke(i)){
            block.invoke(i)
            i++
        }
    }
}

```

```

inline fun tryCatch(tag: String = "Tag", block: () -> Unit){
    try {
        block()
    }catch (e: Exception){
        println("TryCatchException $tag")
        e.printStackTrace()
    }
}

```

```

infix fun <T: R, R> T.isOneOf(types: List<Class<out R>>): Boolean {
    val contextClass = this!!::class.java
    types.forEach {
        if (it == contextClass){
            return true
        }
    }
    return false
}

```

```

package com.alterjuice.utils

```

```

import kotlinx.coroutines.Job
import kotlinx.coroutines.cancelChildren
import kotlinx.coroutines.runBlocking

```

```

class RenewableJob constructor(private var creator: (() -> Job)? = null) {
    private var job: Job? = null

    fun applyOnJob(block: Job.() -> Unit){
        job?.apply(block)
    }
}

```

```

fun cancelIt() {
    applyOnJob {
        cancelChildren()
        cancel()
    }
    job = null
}
fun clearAll(){
    cancelIt()
    this.creator = null
}

fun renewWith(creator: (() -> Job)? = null){
    this.creator = creator
    renew()
}

fun renew() {
    cancelIt()
    job = this.creator?.invoke()
}
}

package com.alterjuice.utils

interface HandlerHolder<T> {
    fun removeHandler(key: String){ }
    fun getHandler(key: String): T?{ return null }
    fun addHandler(key: String, handler: T)
}

```

ДОДАТОК В
Слайди презентації



Рисунок В.1 – Слайд 1



Рисунок В.2 – Слайд 2

Задачі роботи

При написанні дипломної кваліфікаційної роботи було встановлено такі задачі для успішної розробки застосунку:

- Аналіз предметної області та аналогів розроблюваного продукту
- Обрати платформу, для якої буде розроблюватись продукт
- Обрати мову програмування для розробки
- Вибір інструментів та середовищ для розробки застосунку
- Розробити застосунок притримуючись порад при написанні коду
- Провести тестування застосунку на різних пристроях та версіях операційної системи що було обрано

Рисунок В.3 – Слайд 3

Мета роботи, об'єкт та предмет дослідження

- Мета мобільного додатку – це надання можливості користувачам створювати мережеві групи, підключатися до таких груп та підтримувати комунікацію зі всіма учасниками групи, з легким для розуміння інтерфейсом.
- Об'єкт дослідження – програмні засоби для підтримки зв'язку та спілкування.
- Предмет дослідження – застосунок призначений для комунікування без доступу до мережі інтернет.
- Галузь використання – створення груп для спілкування між користувачами у межах діапазону роботи Wi-Fi у разі необхідності зв'язку, але відсутності або нестабільності Інтернет з'єднання.

Рисунок В.4 – Слайд 4

Порівняльна таблиця застосунків-аналогів

Критерії оцінки	Назва застосунку			
	Bridgefy	Briar	Talkie	Juicy Wi-Fi Chat
Інтуїтивна зрозумілість	-	+	+++	+++
Зручність	±	+	++	+++
Сучасність дизайну	++	+	-	++
Споживання ОЗУ	Від 70 МБ	Від 60 до 100 МБ	Від 90 МБ	Від 40 до 100 МБ
Споживання електроживлення	≈100 mAh	≈180 mAh	≈220 mAh	≈180 mAh
Стабільність роботи	-	++	+	++

Рисунок В.5 – Слайд 5

Основні функції застосунку

- Створення мережеских кімнат
- Пошук мережеских кімнат із користувачами поблизу, можливість з'єднання із такою кімнатою.
- Можливість спілкування за допомогою обміну текстових повідомлень із всіма учасниками кімнати.
- Можливість перегляду історії спілкування
- Можливість редагування даних кімнати (серверу) та профілю

Рисунок В.6 – Слайд 6

Вигляд навігації та зв'язків між вікнами



Рис. 1 – зв'язки-шляхи переходів між вікнами застосунку

Рисунок В.7 – Слайд 7

Рис. 2 – Кнопка початку сканування серверів

Інтерфейс застосунку

Застосунок має одну основну кольорову гаму – відтінки та суміші синього і білого, а дизайн виконано за правилами «Material Design».

Juicy Wi-Fi chat
Welcome, Богдан

Рис. 3 – Текст-привітання із обраним іменем користувача

Рис. 4 – перегляд профілю

Рисунок В.8 – Слайд 8

Інтерфейс застосунку. Форма редагування даних серверу.



Рис. 5 – Кнопка зміни статусу серверу



Рис. 6 – Поля для редагування даних серверу



Рис. 7 – екран редагування даних серверу

Рисунок В.9 – Слайд 9

Інтерфейс застосунку. Форма перегляду результатів сканування



Рис. 8 – список знайдених мережних кімнат поблизу

Результати додаються до списку у режимі реального часу.

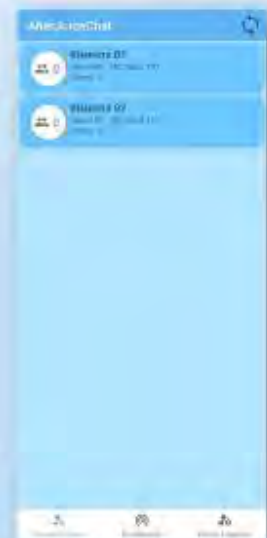


Рис. 9 – Перегляд результатів сканування

Рисунок В.10 – Слайд 10

Інтерфейс застосунку. Вікно із списком доступних чатів



Рис. 10 – список користувачів, що приєдналися до однієї кімнати

Поле чату містить інформацію про кількість непрочитаних повідомлень, текст останнього повідомлення та статусу «онлайн» користувача.



Рис. 11 – вікно зі списком користувачів

Рисунок В.11 – Слайд 11

Інтерфейс застосунку. Форма спілкування із користувачем



Рис. 12 – перегляд форми спілкування

Було вирішено додати альтернативний колір повідомлень співрозмовника – зелений. Він відмінно вписується у загальну кольорову гаму.

Рисунок В.12 – Слайд 12

Інтерфейс застосунку. Вікно налаштувань застосунку

- «**Clear all data**» - очищує збережені дані користувача.
- «**About app**» - при натисканні відкривається діалогове вікно з інформацією про розробника застосунку.

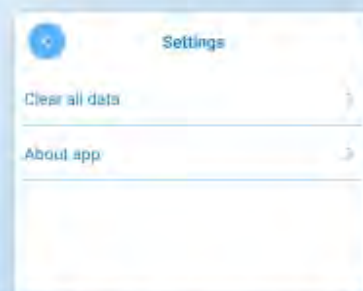


Рис. 14 – перегляд налаштувань

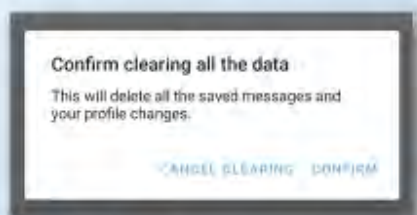


Рис. 13 – Підтвердження видалення даних



Рис. 15 – діалогове вікно з інформацією про розробника

Рисунок В.13 – Слайд 13

Інтерфейс застосунку. Інтерактивні повідомлення



Рис. 16 – повідомлення інформують користувача про роботу сервісу пошуку мережових кімнат та інформацію про те, що особистий сервер працює у даний час.

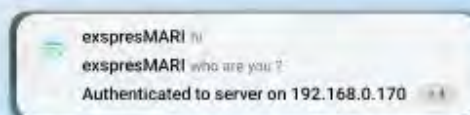


Рис. 17 – згруповані текстові повідомлення від користувачів

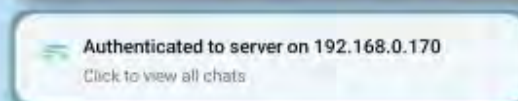
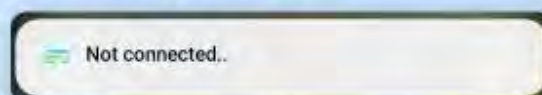


Рис. 18 – Статуси «Не підключений» та «Авторизований»

Рисунок В.14 – Слайд 14

Кастомний елемент - WiFiWavesView

Для більш приємної та живої роботи застосунку створено кастомний елемент інтерфейсу – **анімовані хвилі**, що за дизайном розміщуються за кнопкою круглої форми. Ці хвилі **символізують роботу модуля Wi-Fi**.



Рис. 19 – Вигляд розробленого елемента WiFiWavesView



Рис. 20 – Вигляд елемента WiFiWavesView у застосунку

Рисунок В.15 – Слайд 15

Тестування застосунку на різних пристроях

Критерії тестування	Назва пристрою та версія Android		
	Honor 7A – 8.1	Планшет T1001XS – 8.0	Xiaomi Mi10T – 12.0
Плавність анімації та роботи застосунку	Іноді анімація підлагує	Відмінно	Відмінно
Перевірка коректного завантаження	Коректно	Коректно	Коректно
Правильність та вчасність обробки статусів з'єднання	Відмінно	Відмінно	Відмінно
Перевірка повідомлень	Коректно	Коректно	Коректно
Коректність роботи сервісів (зв'язку із сервером користувача)	Коректно	Коректно	Коректно
Адаптивність інтерфейсу	Місцями - ні	Так	Так

Рисунок В.16 – Слайд 16

Висновки

Під час розробки застосунку було оновлено знання з проектування архітектури MVVM, переп'януті існуючі патерни проектування.

Також було розроблено такі компоненти системи Android та використано найсучасніші бібліотеки:

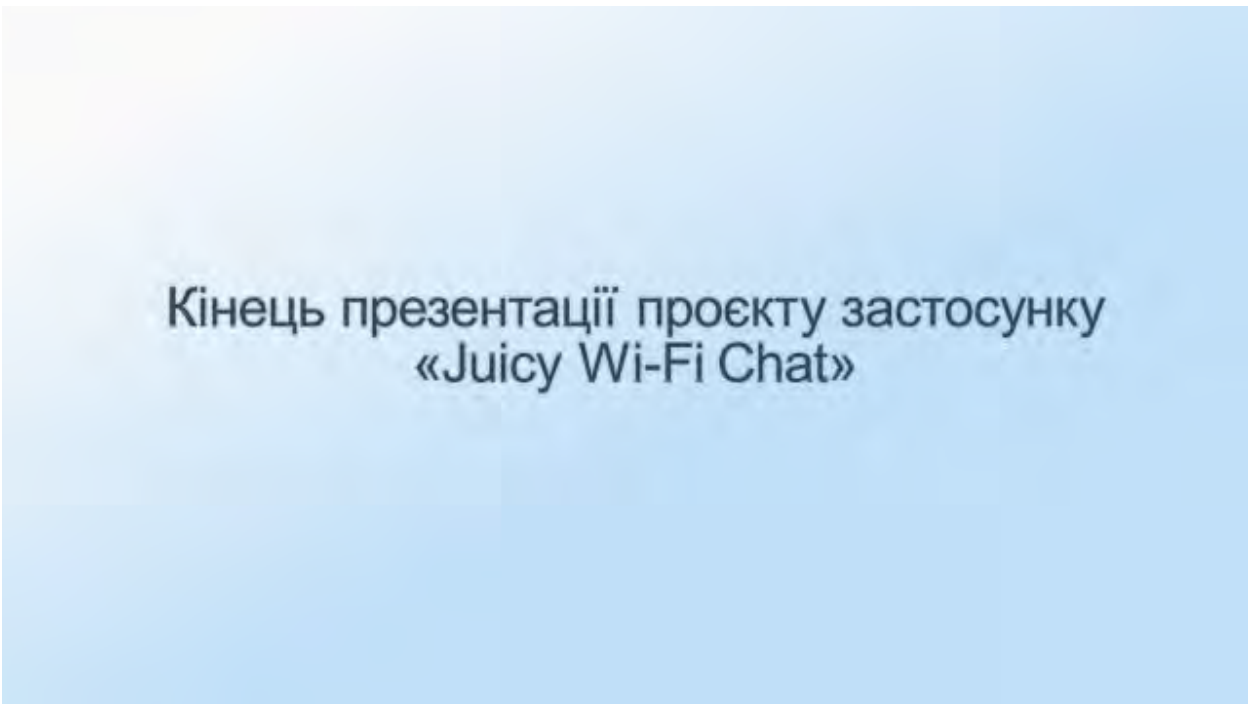
- Розробка особистих сервісів, що можуть працювати у фоні.
- Написання алгоритмів з'єднання та обміну даними між користувачами.
- Вдосконалено навички програмування на мові Kotlin.
- Використано NavigationUI Component, що дозволяє легко переміщуватись між вікнами застосунку.
- Отримано навички з написання кастомного елемента інтерфейсу, а саме – WiFiWavesView.

Рисунок В.17 – Слайд 17

Шляхи вдосконалення функціональності застосунку

1. Міграція механізму збереження даних, втілення сучасних інструментів для роботи із базами даних.
2. Підтримка відправлення медіа файлів користувачам.
3. Підтримка групових чатів.
4. Підтримка вибору та відображення фотографії користувача у чатах.
5. Шифрування даних.

Рисунок В.18 – Слайд 18



Кінець презентації проєкту застосунку
«Juicy Wi-Fi Chat»

Рисунок В.19 – Слайд 19