

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему МОБІЛЬНИЙ ДОДАТОК ДЛЯ СПІЛКУВАННЯ ЗА ДОПОМОГОЮ
ТЕХНОЛОГІЇ BLUETOOTH. MOBILE APPLICATION FOR
COMMUNICATION VIA BLUETOOTH TECHNOLOGY

Виконав: студент(ка) 4 курсу, групи КНТ-127
Спеціальності 121 Інженерія програмного
забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Євдокімов Я. А.

(прізвище та ініціали)

Керівник Степаненко О. О.

(прізвище та ініціали)

Рецензент Голуб Т. В.

(прізвище та ініціали)

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Степаненко О. О., доцент		
Нормоконтроль	Камінська Ж. К., асистент		

7. Дата видачі завдання “1” лютого 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5–6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи.	9 тиждень	

Студент (ка) _____ Євдокімов Я.А.
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ Степаненко О.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
93с., 27 рис., 4 табл., 3 додатки, 17 джерел

МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, BLUETOOTH,
СПІЛКУВАННЯ, JAVA, SQL, JSON, ANDROID STUDIO

Об'єкт дослідження - програмні засоби для організації спілкування за допомогою технології Bluetooth.

Предмет дослідження - мобільний застосунок для реалізації організації спілкування за допомогою технології Bluetooth.

Мета роботи - створення мобільного застосунку, що надає змогу користувачам приєднуватися до пристрою один одного та спілкуватися, використовуючи функції, що надає технологія Bluetooth.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування на мові Java, мобільний пристрій під управлінням операційної системи Android версії 5 або вище.

Результати. Створено мобільний застосунок, який за допомогою функцій технології Bluetooth допомагає користувачам: знаходити пристрої один одного, виконувати з'єднання цих пристроїв, надає доступ до спілкування.

Висновки. Розроблено мобільний застосунок, що допомагає користувачам спілкуватися без доступу до глобальної мережі Ethernet.

Галузь використання - середа, у якій користувачі не мають доступу до інтернету, але мають необхідність у спілкуванні за допомогою мобільного пристрою.

ABSTRACT

Explanatory note to the qualifying work of the bachelor: 93 pages, 27 figures, 4 tables, 3 appendixes, 17 sources

MOBILE APPLICATION, ANDROID, BLUETOOTH, SPEAKING, JAVA, SQL, JSON, ANDROID STUDIO

Object of study is a software for organization communication using the Bluetooth technology.

Subject of study is a mobile app that implements functions of communication using the Bluetooth technology.

The purpose of the work is to create the mobile app that allow users to perform connecting to another device and speaking using the default Android Bluetooth technology.

Metarials, methods, and tools: OOP-programming on Java, mobile device (smartphone) that powered by Android OS (version 5 or higher).

Results. Android application is created that using default Android Bluetooth functions to allow users perform communication.

Conclusions. Android application is created that helps users to communicate without the access to global network.

The field of use is a environment where user do not have access to network, but have a need to communicate.

ЗМІСТ

Перелік скорочень та умовних позначок.....	8
Вступ.....	9
1 Аналіз предметної області	10
1.1 Знайомство з предметною областю.....	10
1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання.....	10
2 Вибір та обґрунтування структури системи, яка проєктується	15
2.1 Вибір інструментарію розробки застосунку	15
2.2 Структура вікон Android застосунку.....	18
2.3 Структура компонентів Android застосунку	21
2.4 Опис модулів та компонентів розробленого застосунку.....	22
3 Основні рішення під час реалізації основних компонентів застосунку, що розробляється	24
3.1 Сервіс BluetoothChat	24
3.2 Форма сканування Bluetooth (ScanListActivity)	29
3.3 Форма (фрагмент) для спілкування (ChatFragment).....	32
3.4 Форма привітання нових користувачів	39
3.5 Форма відрисовки головного користувацького інтерфейсу.....	42
3.6 База даних	45
3.7 Допоміжні класи	53

4 Експлуатація тестування та експериментальне дослідження програми.....	54
4.1 Діаграма послідовностей.....	54
4.2 Тестування застосунку.....	55
Висновки.....	58
Перелік джерел посилань	59
Додаток А Технічне завдання.....	61
Додаток Б Текст програми.....	65
Додаток В Слайди презентації.....	85

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface

JSON – JavaScript Object Notation

NDK – Native Development Kit

RFCOMM – Radio Frequency Communication

SDK – Software Development Kit

SMS – Short Message Service

SQL – Structured Query Language

TCP – Transmission Control Protocol

UUID – Universally Unique Identifier

ВСТУП

Спілкування - один з найважливіших аспектів у виживанні людини в соціумі. Коли для підтримки фізіологічного стану необхідно їсти та пити, для підтримки доброго психологічного стану людині необхідно спілкуватися, як би хто не казав, що може обходитись і без цього.

Не дивлячись на дуже швидку тенденцію розвитку технологій сучасного світу, у тому числі й технологію зв'язку однієї людини з іншою, на сьогодні не завжди є стабільний доступ до глобальної мережі інтернет, отже не завжди є змога просто достати смартфон та відправити повідомлення до свого друга.

Мобільний застосунок, що було розроблено у деякому сенсі вирішує цю проблему, надаючи доступ до відправки повідомлень без підключення до інтернету - за допомогою технології Bluetooth, але також має свої мінуси: за допомогою Bluetooth не можливо спілкуватися з людиною, що, наприклад, знаходиться у іншому місті.

Також основною перевагою вважається те, що цей застосунок є максимально простим у використанні, нема необхідності вивчати програмний інтерфейс перед тим, щоб відправити комусь повідомлення - все зрозуміло с першого погляду.

З'єднатися можна з будь-яким пристроєм, що має технологію Bluetooth та працює під управлінням операційної системи Android. Функціонал застосунку схожий з тим, що надає служба коротких повідомлень (SMS), але виконує це безкоштовно [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Знайомство з предметною областю

Навідміну від нескічено великої кількості сучасних застосунків, що надають користувачам доступ до спілкування, вони працюють тільки при наявності стабільного доступу до Інтернету.

Основна функція застосунку - передача та прийом повідомлень, виконання пошуку пристроїв поруч, виконання з'єднання пристроїв.

1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

З розвитком технологій з'являється ще більше і більше застосунків, що виконують аналогічні функції, та працюють на майже одних й тих самих технологіях. Але, якщо дивитися ринок застосунків, що використовують технологію Bluetooth, то кількість застосунків одразу зменшується у декілька разів.

Назва застосунку, що буде розроблено: “Yorick Messenger” або скорочено: “Y.Messenger”.

1.2.1 Застосунок “Bluetooth Chat”

Як приклад гідного застосунку, що використовує технології Bluetooth можна зазначити застосунок під назвою “Bluetooth Chat”. Розроблений на мові програмування Kotlin, надає доступ до усього, що потрібно користувачу, та має більш менш привабливий дизайн. Інтерфейс локалізований. Головне меню зображено на рисунку 1.1 [2].

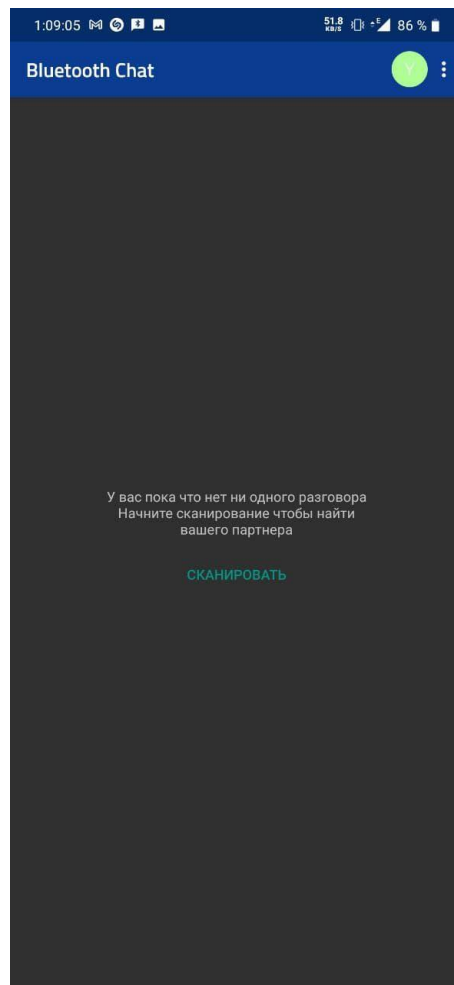


Рисунок 1.1 – Головне меню застосунку “Bluetooth Chat”

1.2.2 Застосунок “BluetoothChat”

Наступний аналог, що також має назву “BluetoothChat” розроблено під старі версії Android, має достатній для розуміння програмний інтерфейс і надає користувачеві доступ до відправки текстових повідомлень. Приклад інтерфейсу наведено на рисунку 1.2 [3].



Рисунок 1.2 - Інтерфейс застосунку “BluetoothChat”

1.2.3 Застосунок “BlueChat”

Застосунок “BlueChat”, що як і перший аналог розроблений на мові програмування Kotlin має достатній функціонал для звичайного обміну повідомленнями та інтерфейс у сучасному стилі. Приклад наведено на рисунку 1.3 [4].

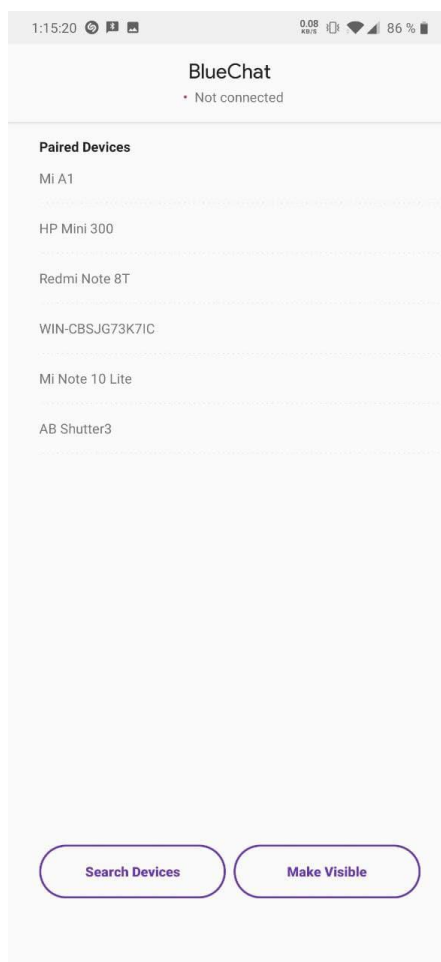


Рисунок 1.3 – Приклад інтерфейсу застосунку “BlueChat”

1.2.4 Порівняльна таблиця програмних продуктів

Розглянуті аналоги мають свої недоліки та свої переваги. Деяки мають більш привабливий інтерфейс, але виконують менше функцій, а деяки навпаки – мають не дуже привабливий вид, але можуть надати користувачу доступ до достійно корисного функціоналу. Порівняльна характеристика застосунків наведена на таблиці 1.1.

Таблиця 1.1 - Порівняльна характеристика застосунків

Назва	Привабливість та зручність інтерфейсу	Стабільність	Зрозумілість
Bluetooth Chat	+	+	+
BluetoothChat	-	+	+
BlueChat	+	-	+
Yorick Messenger	+	+	+

2 ВИБІР ТА ОБГРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЄКТУЄТЬСЯ

2.1 Вибір інструментарію розробки застосунку

2.1.1 Операційна система Android та Android Studio

Так як застосунок, що розробляється, повинен працювати на мобільних пристроях під управлінням операційної системи Android, єдиний вибір, що можна було зробити - це обрати середу розробки Android Studio та операційну систему Android.

Android — операційна система і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на базі ядра Linux. Підтримується альянсом Open Handset Alliance (ОНА) [5].

Одна з проблем розробки під Android полягає у тому, що є велика кількість його версій, та у кожній версії API постійно змінювалось, тому зворотня підтримка старих версій може бути під великим питанням. Існуючі версії Android наведено у таблиці 2.1 [6].

Таблиця 2.1 - Версії Android

Назва	Номер	Рівень використовуваного API	Рівень NDK
1	2	3	4
Cupcake	1.5	3	1
Donut	1.6	4	2
Eclair	2.0-2.1	5	
	2.0.1	6	
	2.1	7	3
Froyo	2.2.x	8	4
Gingerbread	2.3-2.3.2	9	5
	2.3.3-2.3.7	10	
Honeycomb	3.0	11	

Продовження таблиці 2.1

1	2	3	4
Honeycomb	3.1	12	6
	3.2		
Ice Cream Sandwich	4.0.1-4.0.2	14	7
	4.0.3-4.0.4	15	
Jelly Bean	4.1-4.3	16-18	8
KitKat	4.4-4.4.4	19	
Lollipop	5.0-5.1	21-22	
Marshmallow	6.0	23	
Nougat	7.0-7.1	24-25	
Oreo	8.0-8.1	26-27	
Pie	9	28	
Android 10 (Q)	10	29	
Android 11 (R)	11	30	

2.1.2 Мова програмування

Перед початком розробки було розглянуто декілька мов програмування, що використовуються для розробки застосунків під операційну систему Android.

Таким чином, на вибір було взято: Java, Kotlin, Dart (фреймворк Flutter) та C# (фреймворк Xamarin) [7].

Було обрано мову програмування Java, як класичну мову для програмування застосунків під операційну систему Android.

Kotlin - це відносно нова мова, яка зараз швидко набирає популярність, та в деяких аспектах вважається зручнішою за Java. [8], [9]

Dart- це нова мова програмування, що використовується у фреймворкі Flutter від Google, відрізняється від вище зазначених своєю кроссплатформенністю: написаний застосунок мовою програмування Dart

може бути без проблем одночасно встановлений та запущений на Android, iOS та на комп'ютері у вигляді сайту [10].

Xamarin – це платформа з відкритим вихідним кодом, що надає розробникам доступ до кросплатформеного програмування мобільних застосунків за допомогою .NET.

2.1.2 Огляд особливостей середовища розробки

Як було зазначено вище - було обрано середовище розробки Android Studio, що є зараз максимально зручною під задачі, що для неї ставляться.

Android Studio – дуже знайома всім мобільним розробникам середовище, яке не просто так завоювало свою репутацію, бо надає дійсно зручний інструментарій для проектування застосунків [11]:

- засновано на розробках IntelliJ IDEA від JetBrains, що свідчить про доступ до магазину плагінів JetBrains;
- має редактор макетів WYSIWYG;
- компілювання проектів за допомогою Gradle;
- рефакторинг коду;
- підтримка великої кількості платформ, що працюють під операційною системою Android (наприклад, Wear OS та Android TV);
- підтримка Google Cloud;
- аналізатор кода Lint;
- збірка .apk файлів різної конфігурації.

Порівняльні характеристики існуючих середовищ розробки наведено у таблиці 2.2.

Таблиця 2.2 – Порівняльна характеристика

Назва	Мова(и) програмування	Платформи розробки	Платна
Android Studio	Java, C/C++, Dart,	Android	Ні
Microsoft Visual Studio (Xamarin)	C#	Android, iOS	Ні
Unreal Engine	C, C++	Android, iOS - пізніше; Windows, PlayStation, Xbox	Ні/Використовує ліцензії
Unity 3D	C#, C++	iOS, Android, macOS, Windows, Linux та ін.	Ні/Використовує ліцензії
Eclipse IDE	Java, PHP, Ruby, Python	Windows, iOS, Android, Linux	Ні
XCode	C/C++, Objective-C, Swift, Java, AppleScript	iOS, OSX	Ні

Android Studio було обрано виходячи з особистого досвіду використання - на мою думку, Android Studio має достатньо зручний інтерфейс та добрий набір інструментів та має добре зроблену документацію.

2.2 Структура вікон Android застосунку

Кожний застосунок, що розробляється, базується на структурі шарів (макетів) (англ. - “layouts”). Кожна “активність” (англ. - “activity”, далі - форма) такого застосунку має містити хоча б один шар - кореневий. Після додавання кореневого шару до форми можна додавати величезну кількість інших шарів

різних типів, створювати “складені” шари, які містять у собі декілька інших [12] - [14].

Можна виділити декілька основних типів:

– `LinearLayout (LL)` – розташовує усі елементи у одному напрямку - горизонтально або вертикально, один за одним. Приклад шару наведено на рисунку 2.1;

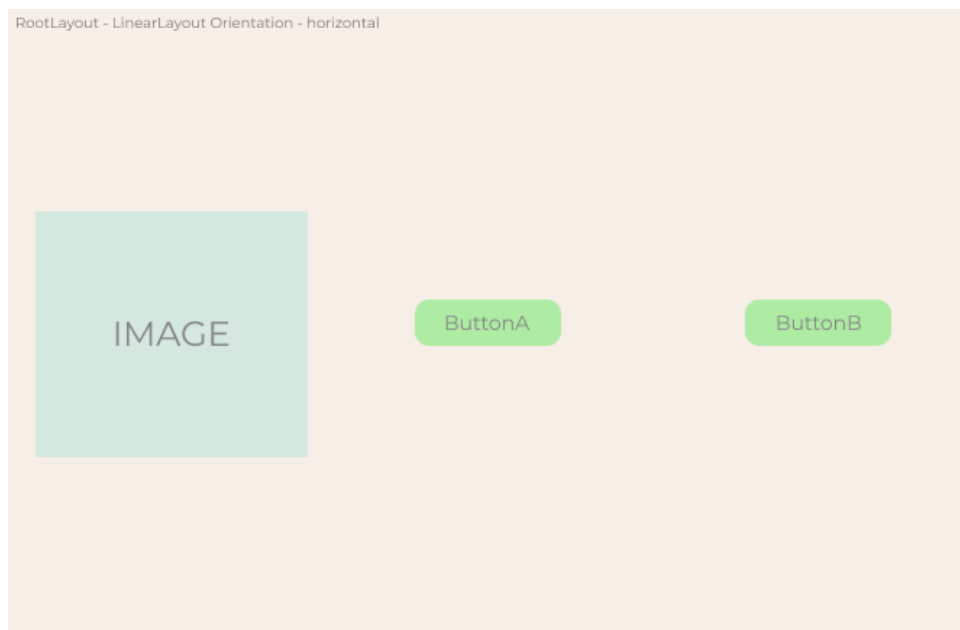


Рисунок 2.1 - Приклад горизонтального розташування елементів у LL

– `RelativeLayout (RL)` – розташовує усі елементи відносно до іншого. Позиції елементів можна зазначати відносно до будь-якого “родича”, наприклад вище його, або зліва. Приклад шару наведено на рисунку 2.2;

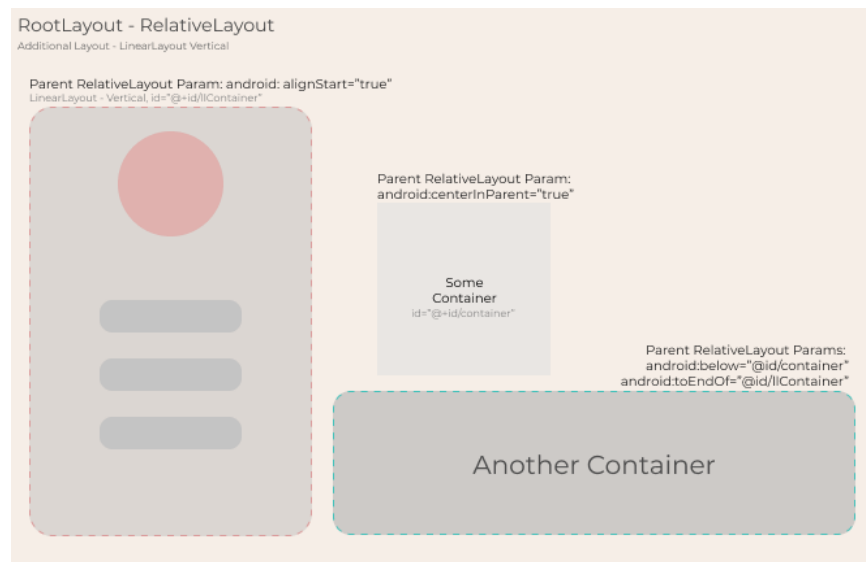


Рисунок 2.2 - Приклад розташування елементів за допомогою RL

– ConstraintLayout (CL) – дозволяє “гнучко” розташовувати елементи за допомогою “якорів”. Кожен елемент повинен мати хоча б один горизонтальний та вертикальний якор. Приклади наведено на рисунку 2.3;

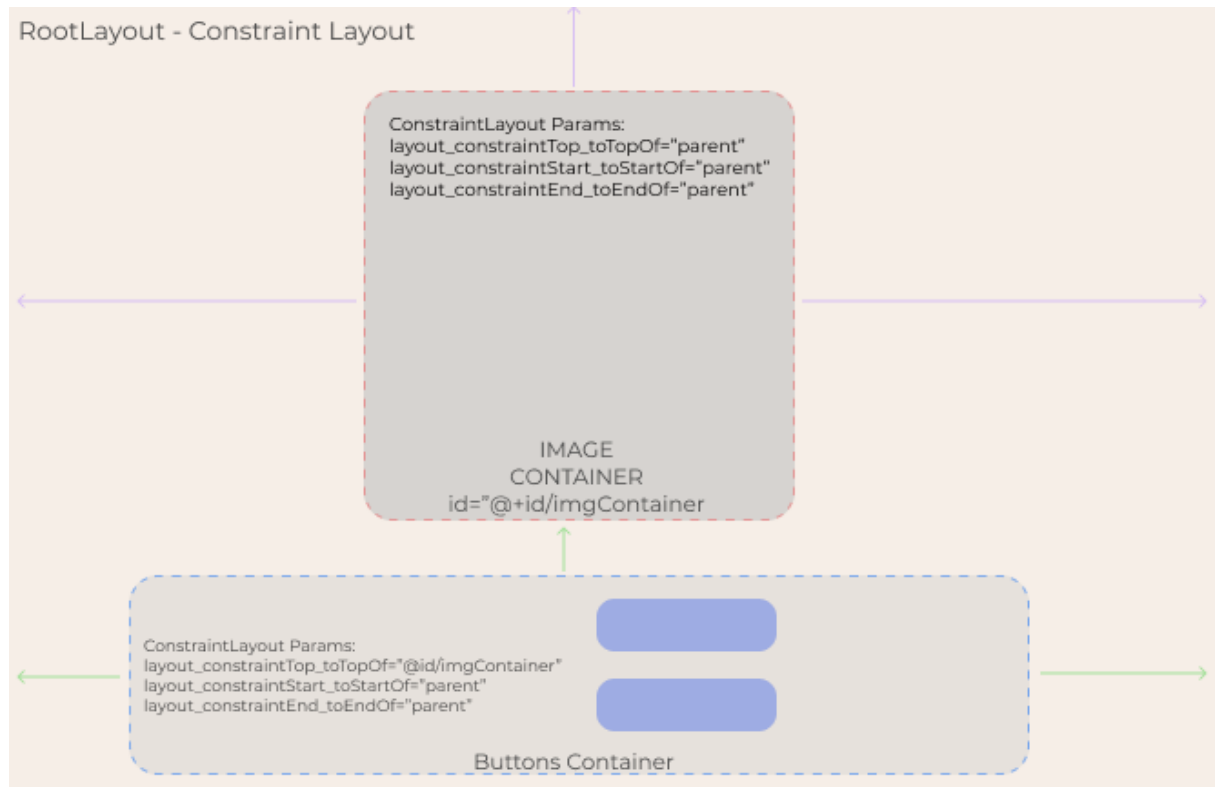


Рисунок 2.3 - Приклад розташування елементів за допомогою CL

- `AbsoluteLayout` - дозволяє розташовувати елементи за координатами `x` та `y`. На сьогодні вважається застарілим та не використовується.

2.3 Структура компонентів Android застосунку

На рисунку 2.4 наведено структуру компонентів та модулів проекту. Головними слід зазначити:

- `manifests` - директорія, що містить у собі файл `AndroidManifest.xml`, який у свою чергу містить у собі інформацію про застосунок: кількість форм, які дозволи необхідно надати застосунку для коректної роботи та ін.;
- `java` - директорія, що містить у собі програмний код для всіх форм;
- `res` - директорія, що містить у собі усі ресурси проекту (зображення, `xml` файли з розміткою форм, анімації, тощо).

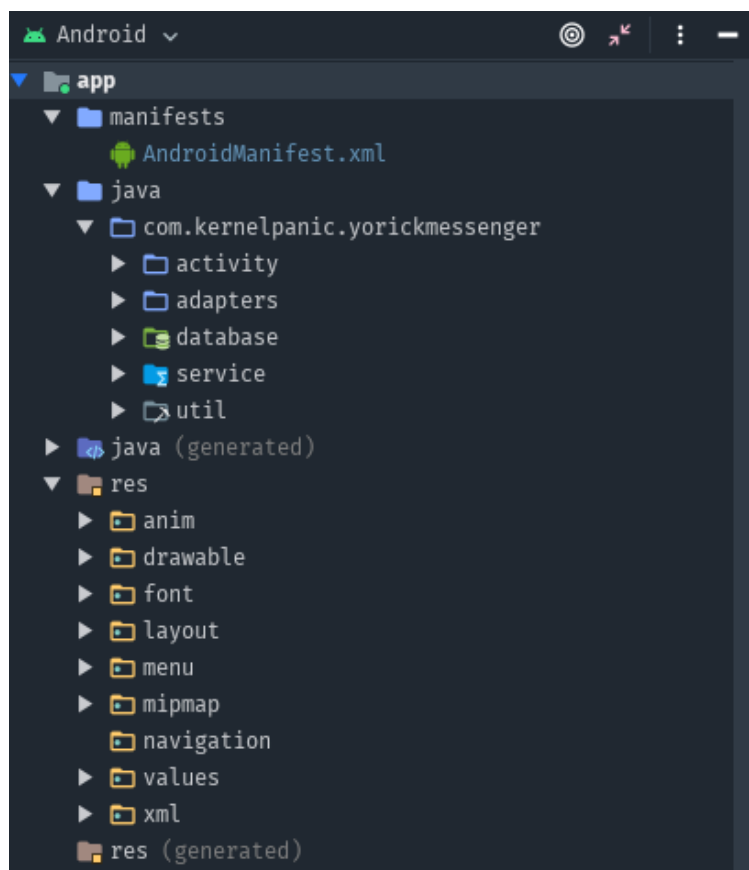


Рисунок 2.4 - Структура компонентів проекту

2.4 Опис модулів та компонентів розробленого застосунку

2.4.1 Класи, модуль Java

Можна виділити декілька основних класів, що розроблені для виконання основних функцій застосунку.

BluetoothChatService - майже найголовніший клас, який виконує основну логіку застосунку - виконує операції з Bluetooth на пристрої користувача (з'єднання, відправка / отримання повідомлень).

ScanListActivity - клас для форми, що виконує пошук пристроїв у діапазоні роботи Bluetooth на пристрої користувача.

ChatFragment - клас для фрагменту, що надає користувачеві доступ до чату.

MainAppActivity - клас для форми, що відрисовує головний користувацький інтерфейс для всього застосунку.

У застосунку реалізовано класи (для роботи з базами використовується SQLite) для бази даних, що зберігається у пам'яті користувацького пристрою. Зберігає повідомлення та інформацію про користувача для обміну цими даними між пристроями після підключення.

Основним класом виступає SQLiteOpenHelper, що містить у собі опис методів для створення таблиць та роботи з ними.

2.4.2 Ресурси, модуль Res

Директорія drawable містить xml файли з описами деяких елементів. Наприклад, скруглений фон для кнопок, градієнти, програмні іконки; та файли зображень. На рисунку 2.5 наведено приклад одного з таких файлів під назвою «app_ui_gradient_sunnymorning.xml», що містить у собі опис градієнтного фону.

Директорія layout зберігає у собі розмітку для форм проекту.

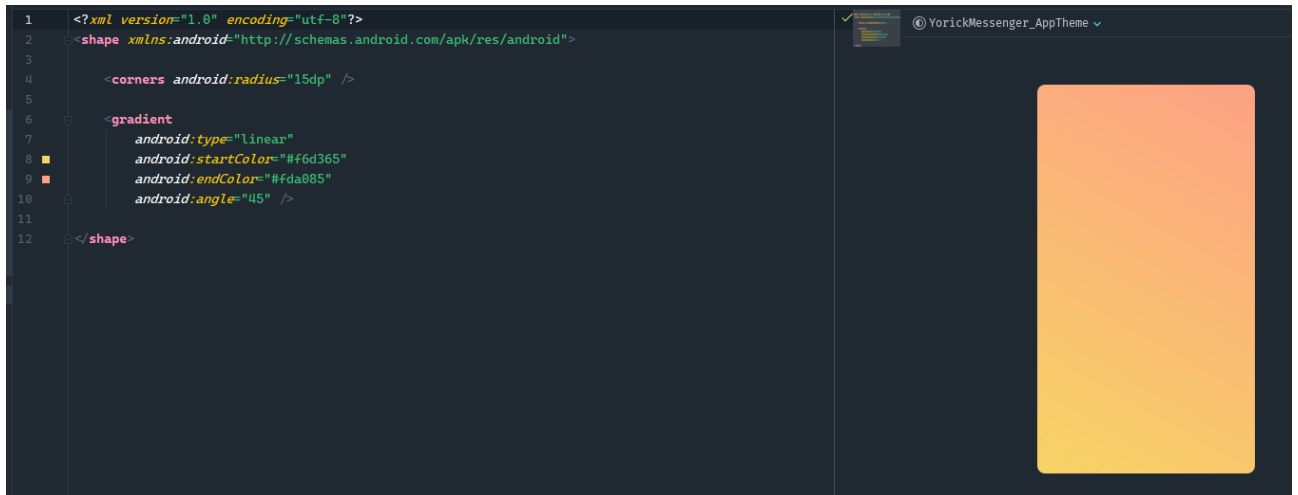


Рисунок 2.5 - Приклад файлу drawable-ресурсу

Директорія values містить xml файли з описом кольорів, розмірів, тем (стилей) для застосунку, строкові найменування, переклади.

3 ОСНОВНІ РІШЕННЯ ПІД ЧАС РЕАЛІЗАЦІЇ ОСНОВНИХ КОМПОНЕНТІВ ЗАСТОСУНКУ, ЩО РОЗРОБЛЯЄТЬСЯ

3.1 Сервіс BluetoothChat

Основою застосунку є клас-сервіс, що реалізує головну логіку Bluetooth-з'єднань. Для того, щоб з'єднати два пристроя використовується клас BluetoothSocket (BluetoothServerSocket).

Інтерфейс BluetoothServerSocket схожий з інтерфейсами TCP сокетів: Socket та ServerSocket. Найбільш поширеним типом сокету є RFCOMM.

Для коректної роботи з віддаленими пристроями Bluetooth необхідно реалізувати наступні підкласи: AcceptThread, ServerThread, ReadWriteThread, які виконують функції приймання вхідних підключень, запуск серверу для прослуховування вхідних підключень та роботу з даними (відправка/отримання). Кожен з цих підкласів розширюється класом Thread, тому є необхідність унаслідувати та перевизначити декілька методів класа-родича.

Для того, щоб зробити пристрій-сервер готовим до приймання з'єднання необхідно виконати BluetoothAdapter.listenUsingRfcommWithServiceRecord() та передати як параметр унікальний ідентифікатор (UUID), що має бути однаковим як на пристрої-сервері, так і на пристрої-клієнті.

Спілкування форми чату та сервісу відбувається за допомогою повідомлень, що буде роздівлено у пункті 3.3.

Схематичний приклад підключення одного пристрою до іншого зображено на рисунку 3.1

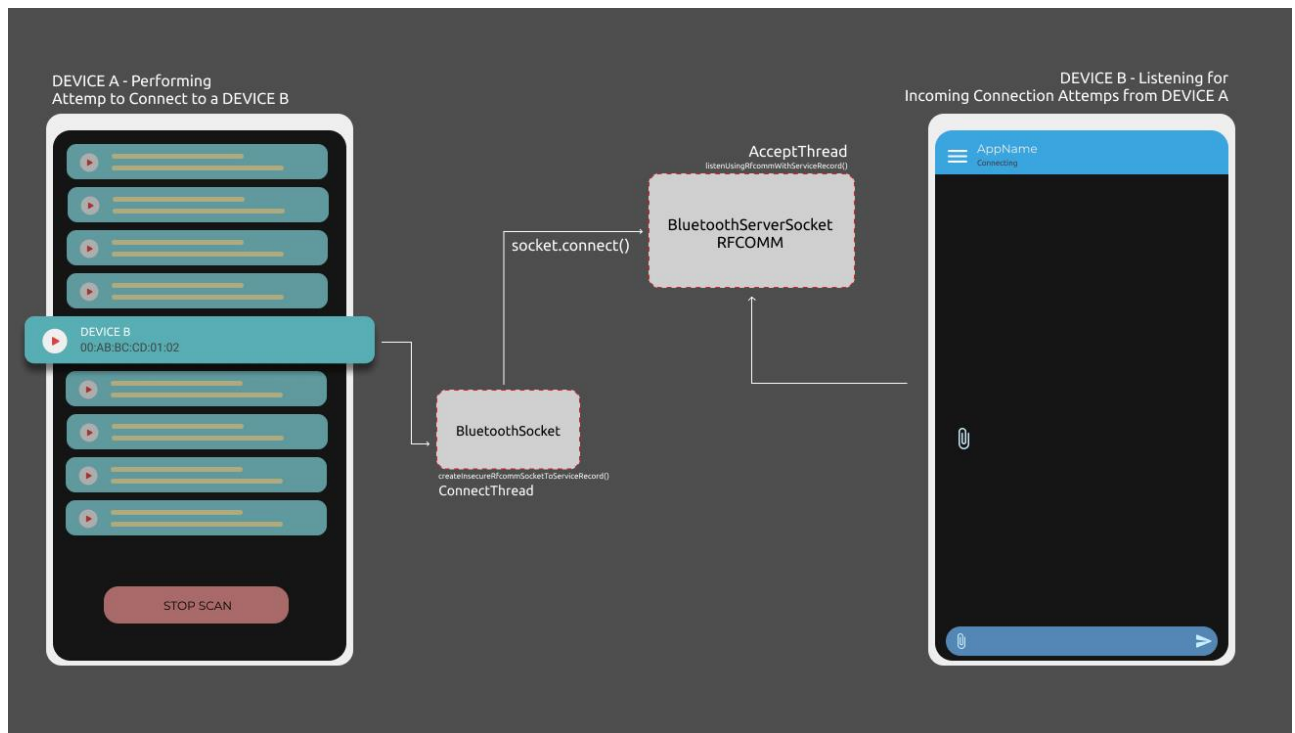


Рисунок 3.1 - Приклад підключення одного пристрою до іншого

Пристрій-сервер (Device B) відкриває сервер для прослуховування на входні підключення (за допомогою класу `AcceptThread`), пристрій-клієнт (Device A) створює новий сокет та виконує спробу підключитися до пристрою-серверу (за допомогою класу `ConnectThread`).

Приклад коду, який демонструє старт серверу:

```
public ServerThread() {
    BluetoothServerSocket tmp = null;

    try {
        tmp =
bluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(app_name, MY_UUID);
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    serverSocket = tmp;
}
```

Приклад коду, який демонструє підключення до серверу:

```
public ConnectThread(BluetoothDevice device) {
    this.device = device;
    BluetoothSocket tmpSocket = null;

    try {
        //tmpSocket = (BluetoothSocket)
device.getClass().getMethod("createRfcommSocket", new Class[]{ int.class
}).invoke(device, 1);
        tmpSocket = device.createInsecureRfcommSocketToServiceRecord(MY_UUID);
    } catch (Exception e) {
        e.printStackTrace();
    }
    socket = tmpSocket;
}
```

Після вдалого підключення запускається клас, який дозволяє двом пристроям обмінюватися даними - перший пристрій відправляє до другого пакет (масив) байтів, другий його приймає та трансформує вже у текстове повідомлення (рис. 3.2), зображення, файл, тощо.

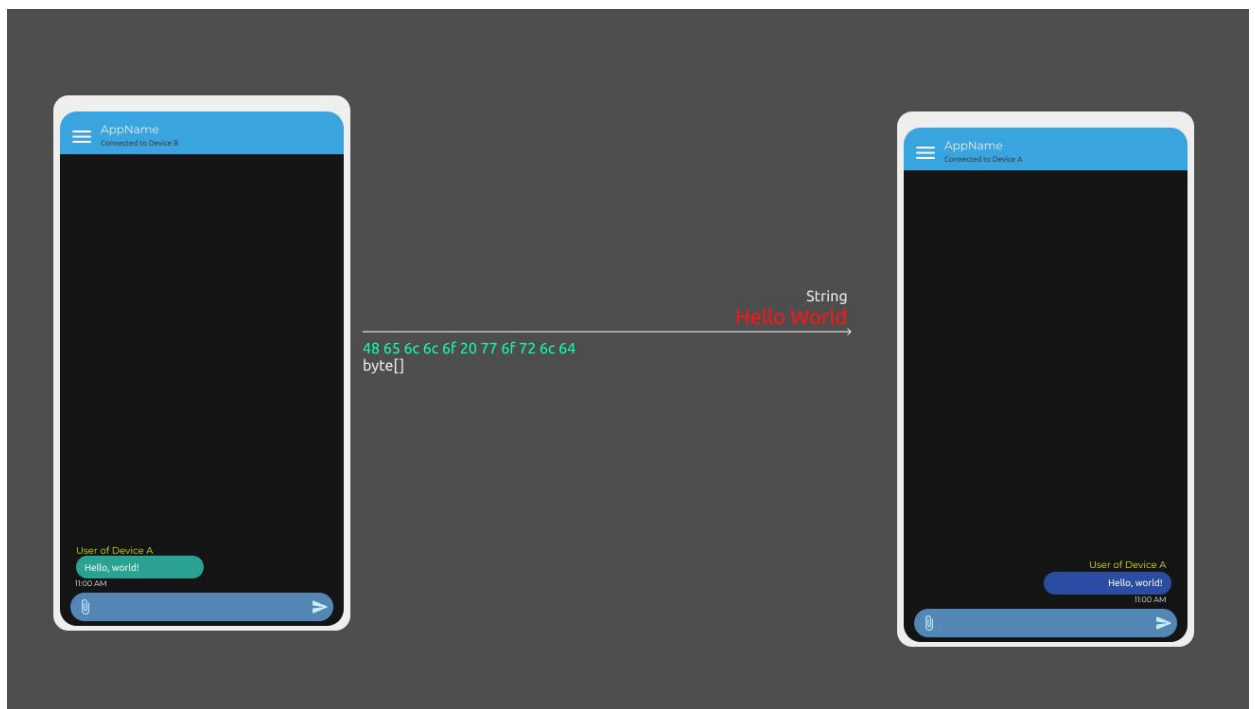


Рисунок 3.2 - Приклад обміну повідомленнями

Приклад коду, який дозволяє виконувати відправку та отримання даних користувачами:

```
switch (dataInputStream.readInt()) {
    case Constants.TYPE_WRITE_DEFAULT:
        String message = dataInputStream.readUTF();
        sendMessageToUI(Constants.MESSAGE_READ, message);
        break;
    (...)
}
try {
    dataOutputStream.writeInt(Constants.TYPE_WRITE_DEFAULT);
    dataOutputStream.writeUTF(message);
} catch (Throwable ex) {
    Log.e(TAG, "We have caught an exception: " + ex);
}
```

Також, у нових версіях Android користувач повинен надати застосунку доступ до Bluetooth та даним о місцезнаходженні.

Для отримання доступу до файлової системи користувача також необхідно отримати дозвіл (READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE).

Методи, які демонструють відправку та отримування файлів за допомогою Bluetooth:

```
public void writeFile(final String filePath) {
    executor.execute(new Runnable() {
        public void run() {
            try {
                sendMessageToUI(Constants.MESSAGE_WRITE_FILE_NOW,
"Sendind file + " + filePath);
                FileInputStream fileInputStream = new
FileInputStream(filePath);
                File file = new File(filePath);
                dataOutputStream.writeInt(Constants.TYPE_WRITE_FILE);
                dataOutputStream.writeUTF(file.getName());
                dataOutputStream.writeLong(file.length());
                int r;
```

```

        byte[] bytes = new byte[4 * 1024];
        while ((r = fileInputStream.read(bytes)) != -1) {
            dataOutputStream.write(bytes, 0, r);
        }
        sendMessageToUI(Constants.MESSAGE_WRITE_FILE,
file.getAbsolutePath());
    } catch (Throwable ex) {
        Log.e(TAG, "We have caught the exception: ", ex);
    }
}

});
}

case Constants.TYPE_WRITE_FILE:
    File destinationDirectory = new File(FILE_PATH);
    if (!destinationDirectory.exists())
        destinationDirectory.mkdirs();
    String filename = dataInputStream.readUTF();
    long fileSize = dataInputStream.readLong();
    sendMessageToUI(Constants.MESSAGE_READ_FILE_NOW,
fileSize);

    long size = 0;
    int r;
    byte[] bytes = new byte[4 * 1024];
    FileOutputStream out = new
FileOutputStream(FILE_PATH + filename);
    while ((r = inputStream.read(bytes)) != -1) {
        out.write(bytes, 0, r);
        size += r;
        if (size >= fileSize) {
            break;
        }
    }
    sendMessageToUI(Constants.MESSAGE_READ_FILE,
FILE_PATH + filename);

    break;

```

На рисунку 3.3 наведено діаграму класу BluetoothChatService.

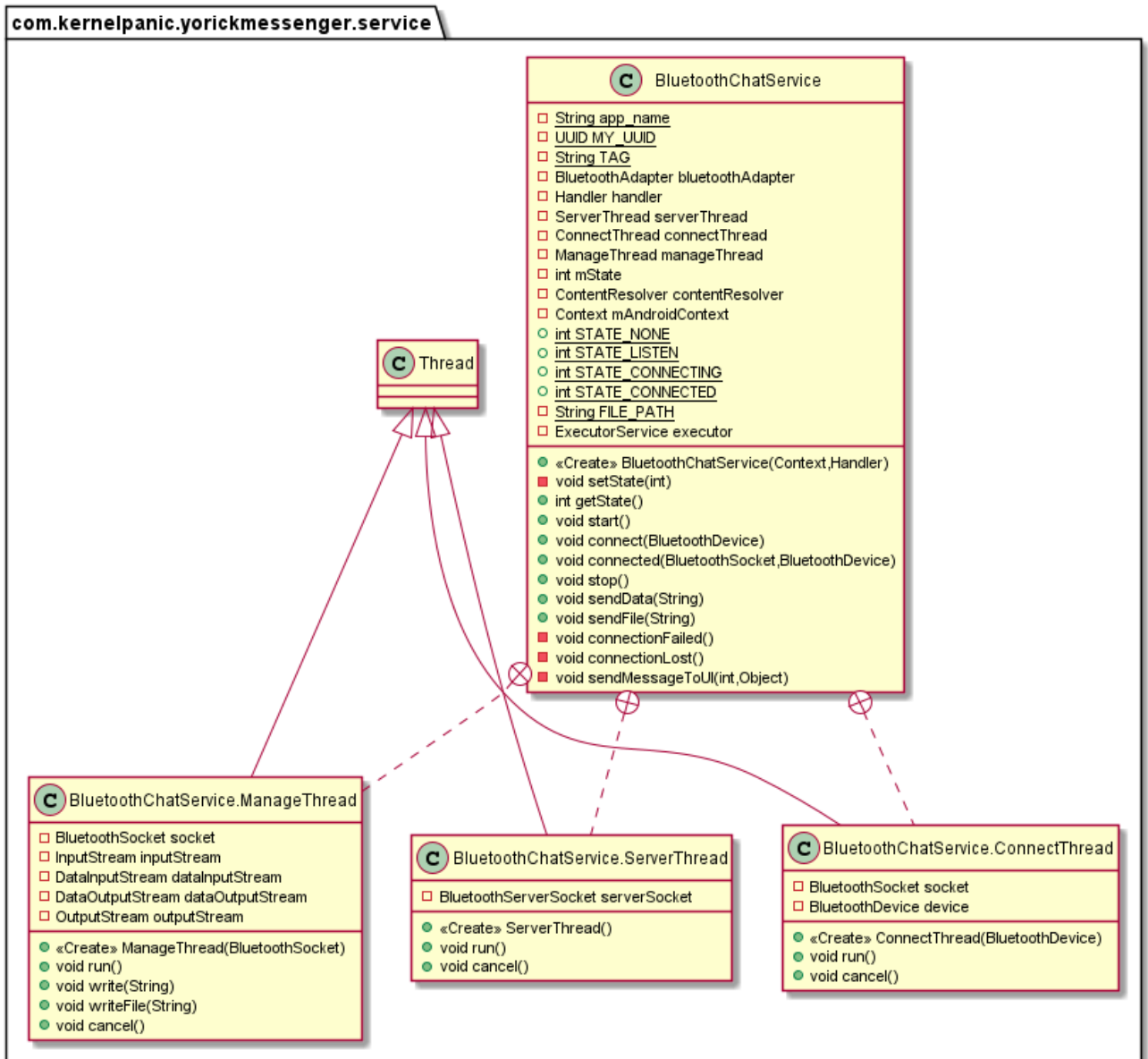


Рисунок 3.3 - Діаграма класу BluetoothChatService

3.2 Форма сканування Bluetooth (ScanListActivity)

Для виконання пошуку інших віддалених пристроїв у зоні досягу Bluetooth було створено форму та відповідний до неї клас, яка надає доступ користувачеві до інтерфейсу пошуку.

Алгоритм пошуку оснований на функціях інтерфейсу BluetoothAdapter який надає доступ до такого методу, як BluetoothAdapter.startDiscovery(). Перед початком роботи реєструються списки paired та non-paired пристроїв:

```

pairedDevicesListAdapter    = new DevicesListAdapter(this,
R.layout.list_single_item);
newDevicesListAdapter      = new ArrayList<Device>();
newDevicesListAdapter      = new DevicesListAdapter(this,
R.layout.list_single_item);

deviceList                  = findViewById(R.id.deviceList);
devicePairedList            = findViewById(R.id.devicePairedList);

deviceList.setAdapter(newDevicesListAdapter);
deviceList.setOnItemClickListener(onDeviceTapListener);

devicePairedList.setAdapter(pairedDevicesListAdapter);
devicePairedList.setOnItemClickListener(onDeviceTapListener);

```

Після цього нам необхідно зареєструвати “дії” - ACTION_%ACTION_NAME%, для цього створюємо об’єкт класу IntentFilter, додаючи до нього необхідні правила та перевизначаєм метод класу BroadcastReceiver:

```

IntentFilter bluetoothActionIntent = new
IntentFilter(BluetoothDevice.ACTION_FOUND);
this.registerReceiver(broadcastReceiver, bluetoothActionIntent);

bluetoothActionIntent = new
IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
this.registerReceiver(broadcastReceiver, bluetoothActionIntent);

private final BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

(...)

    }
}

```

```

    }
};

```

Також, починаючи з Android версії Marshmallow необхідно запитувати у користувача надати доступ застосунку до необхідних функцій, в нашому випадку необхідно надати доступ до Bluetooth (BLUETOOTH, BLUETOOTH_ADMIN) та до даних місцезнаходження (ACCESS_COARSE_LOCATION, ACCESS_FINE_LOCATION) для того, щоб застосунок зміг виконувати пошук.

Після того, як усі підготовки були виконані запускається функція пошуку:

```

private void startDiscovery() {
    if (mBluetoothAdapter.isDiscovering()) {
        mBluetoothAdapter.cancelDiscovery();
    }
    mBluetoothAdapter.startDiscovery();
}

```

Кожний знайдений пристрій додається до списку за допомогою вище описаного методу класу BroadcastReceiver, спарені пристрої додаються до відповідного списку відразу після старту форми (метод onCreate).

Після вибору пристрою до якого користувач бажає підключитися форма закривається, передаючи результат, що все добре (RESULT_OK) до фрагменту з чатом та записує до Bundle адрес обраного пристрою за допомогою методу класу Intent.putExtra(key, value).

На рисунку 3.4 наведено діаграму класу (ScanListActivity) для форми сканування:

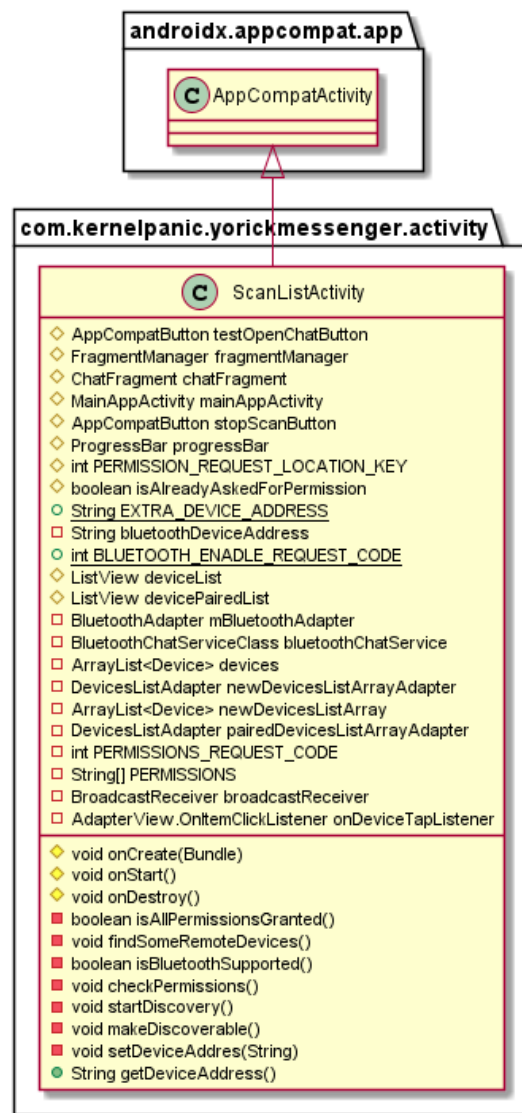


Рисунок 3.4 - Діаграма класів форми сканування

3.3 Форма (фрагмент) для спілкування (ChatFragment)

Цей фрагмент одразу після своєї появи й запускає форму сканування, та очікує результат від неї. Фрагмент починає свою роботу тільки після отримання повідомлення від форми сканування, що все добре і зараз виконується підключення:

```

(...)
setupChat();
Toast.makeText(getActivity(), "Initiating the scan devices procedure",
Toast.LENGTH_SHORT).show();

```

```

Intent server = new Intent(getActivity(), ScanListActivity.class);

Handler postDelayedHandler = new Handler();
postDelayedHandler.postDelayed(new Runnable()
    {
        @Override
        public void run() {
            startActivityForResult(server,
PERMISSION_REQUEST_CONNECT_DEVICE_SECURE);
        }
    }, 1000);
}

```

Обробкою отриманих повідомлень-результатів займається метод `onActivityResult()`, що має наступний вигляд:

```

@Override
public void onActivityResult(int reqCode, int resCode, @Nullable Intent data) {
    Bitmap bitmap = null;
    switch (reqCode) {
        case PERMISSION_REQUEST_CONNECT_DEVICE_SECURE:
            if (resCode == Activity.RESULT_OK && chatService.getState() !=
BluetoothChatService.STATE_CONNECTED) {
                connectToDevice(data);
                onConnectDataExchange();
            }
            break;
        case PERMISSION_REQUEST_CONNECT_DEVICE_INSECURE:
            if (resCode == Activity.RESULT_OK && chatService.getState() !=
BluetoothChatService.STATE_CONNECTED) {
                connectToDevice(data);
                onConnectDataExchange();
            }
    }
}
(...)

```

Головну функцію виконує об'єкт класу `Handler`, який отримує та обробляє повідомлення від класу `BluetoothChatService`. При роботі з віддаленим пристроєм цей клас може повертати різні типи повідомлень, наприклад: повідомлення про зміну стану на “ПІДКЛЮЧЕНО”

(STATE_CONNECTED), повідомлення про необхідність відправити дані (MESSAGE_WRITE), повідомлення про необхідність зчитати дані (MESSAGE_READ).

Приклад відправки повідомлень на методі setState(), який приймає як параметр стан, на який треба змінити поточний:

```
private synchronized void setState(int state) {
    Log.i(TAG, "The setState() method called, state " + mState + " is going to -
> " + state);
    mState = state;
    handler.obtainMessage(Constants.MESSAGE_STATE_CHANGE, state, -
1).sendToTarget();
}
```

Повідомлення що містить у собі інформацію про те, що стан змінився, відправляється до основного об'єкту, а він в свою чергу обробляє його та виконує операції, що відносяться до цього стану. Наприклад, отримавши повідомлення про зміну стану на "ПІДКЛЮЧЕНО" виконується операція першого обміну користувацькою інформацією, тобто один пристрій надсилає до іншого ім'я користувача, для того, щоб при спілкування користувачі мали змогу бачити ім'я, яке вони зазначали при створенні профілю, а не назву свого пристрою.

Приклад:

```
private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        switch (msg.what) {
            case Constants.MESSAGE_STATE_CHANGE:
                switch (msg.arg1) {
                    case BluetoothChatService.STATE_CONNECTED:
                        try {
                            (...)
                            Snackbar.make(getView(), "Wait until we done with
transferring the user data", Snackbar.LENGTH_SHORT).show();
```

```

        toolbar = getActivity().findViewById(R.id.appBar);

toolbar.setSubtitle(getString(R.string.app_mainActivity_connected,
connectedDeviceBluetoothName));

        onConnectDataExchange();
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {

(...)

                Snackbar.make(getView(), "Done",
Snackbar.LENGTH_SHORT).show();
            (...)

            }
        }, 1500);
    } catch (IllegalStateException ex) {
        Log.d(TAG, ex.getMessage());
    }
}

```

Для зручності користувачів було розроблена два різні структури відправлених та отриманих повідомлень, приклади наведено на рисунках 3.5 та 3.6. Реалізовано це за допомогою класу ChatRecyclerViewAdapter, який наслідує клас RecyclerView.Adapter.

Для кожного типу повідомлення було створено свій клас “держак” (ViewHolder), що відрисовував структуру повідомлення. Приклад одного з таких класів:

```

public static class SentMessageViewHolder extends RecyclerView.ViewHolder {
    TextView    username;
    TextView    message;
    TextView    timestamp;

    public SentMessageViewHolder(@NonNull View itemView) {
        super(itemView);
        this.username = (TextView) itemView.findViewById(R.id.usernameLabel);
        this.timestamp = (TextView) itemView.findViewById(R.id.timestampLabel);
        this.message = (TextView) itemView.findViewById(R.id.messageLabel);
    }
}

```

Структура також будується на основі отриманих повідомлень від фрагменту, якщо відправити до адаптеру тип повідомлення “відправлене”:

```
messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(writeMessage,
    currentTimeMillisW, Constants.MESSAGE_TYPE_SENT, sentUsername, false));
```

то буде створено нову структуру для віправленого повідомлення:

```
case SENT:
Log.d("Y.Messenger", "Sent View Holder Created");
view =
LayoutInflater.from(parent.getContext()).inflate(R.layout.message_sent,
    parent, false);
return new SentMessageViewHolder(view);
```

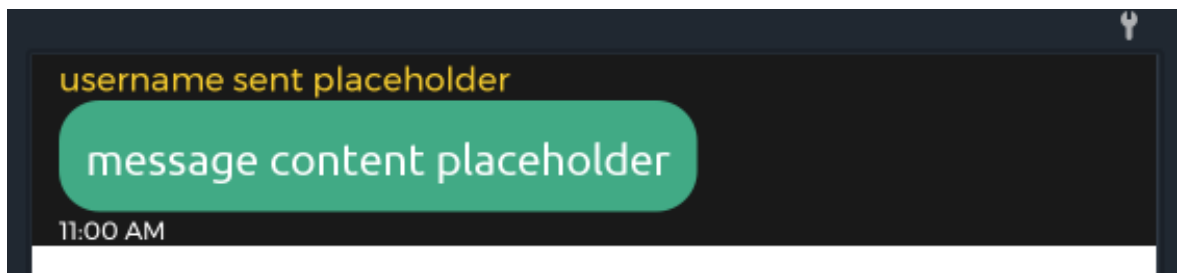


Рисунок 3.5 - Структура відправленого повідомлення

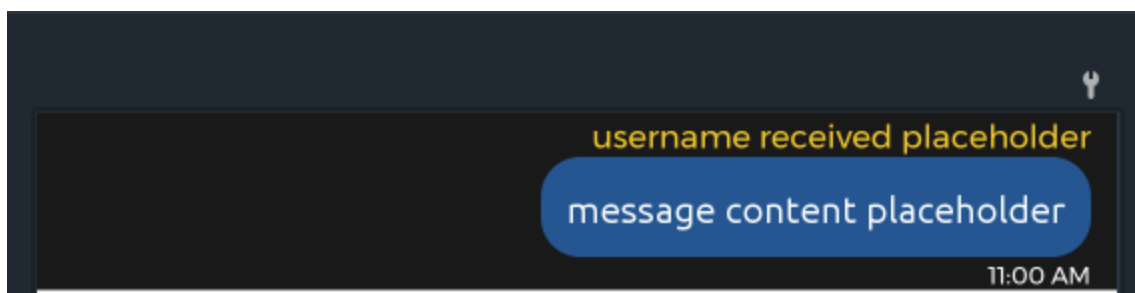


Рисунок 3.6 - Структура отриманого повідомлення

Приклад створення та присвоювання адаптеру до листу повідомлень:

```
LinearLayoutManager linearLayoutManager = new  
LinearLayoutManager(getActivity());  
  
messageList = new ArrayList<com.kernelpanic.yorickmessenger.util.Message>();  
  
chatView = view.findViewById(R.id.messagesContainer);  
chatView.setLayoutManager(linearLayoutManager);  
chatListAdapter = new ChatRecyclerAdapter(messageList, getActivity());  
chatView.setAdapter(chatListAdapter);
```

На рисунку 3.7 наведено діаграму класу (ChatFragment) для фрагменту чата:



Риснок 3.7 - Діаграма класів для форми чату

3.4 Форма привітання нових користувачів

Якщо користувач перший раз запускає застосунок, то йому буде продемонстровано форму, яка розповідає про нього. Реалізовано за допомогою додавання та зчитування записів `SharedPreferences` застосунку.

`SharedPreferences` представляє собою запис виду ключ:значення, яке додається у пам'ять застосунку.

Приклад коду, що перевіряє, чи бачив вже користувач вступну форму:

```
if (checkUserSharedPreferences()) {
    Log.i("YMSNGR_USER_ACT", "User has watched the intro before, launching
ChatActivity");
    Intent chatActivity = new Intent(getApplicationContext(),
MainAppActivity.class);
    startActivity(chatActivity);
    finish();
}
```

Приклад коду додавання запису:

```
SharedPreferences preferences =
getApplicationContext().getSharedPreferences(PREFERENCES_NAME, MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
editor.putBoolean(PREFERENCE_BOOL_KEY, true);
editor.commit();
```

Приклад коду зчитування даних з запису:

```
SharedPreferences preferences =
getApplicationContext().getSharedPreferences(PREFERENCES_NAME, MODE_PRIVATE);
Boolean isUserWatchedIntroActivity = preferences.getBoolean(PREFERENCE_BOOL_KEY,
false);
return isUserWatchedIntroActivity;
```

Мовою за замовченням у застосунку вважається англійська, але на цій формі можна одразу змінити мову на українську за допомогою натискання на кнопку зміни (рисунок 3.8).

Зміна мови також проходить за допомогою SharedPreferences, але тут зберігається значення змінної типу Locale:

```
private String checkLocaleSharedPreferences() {
    String locale = "en";
    SharedPreferences preferences =
getApplicationContext().getSharedPreferences(PREFERENCE_NAME_APPLICATION_LOCALE,
MODE_PRIVATE);
    locale = preferences.getString(PREFERENCE_KEY_APPLICATION_LOCALE, "en");
    return locale;
}

private void setLocale(Activity activity, String localeCode) {
    Locale locale = new Locale(localeCode);
    locale.setDefault(locale);

    Resources resources = activity.getResources();
    Configuration configuration = resources.getConfiguration();
    configuration.setLocale(locale);
    resources.updateConfiguration(configuration, resources.getDisplayMetrics());

    SharedPreferences preferences =
getApplicationContext().getSharedPreferences(PREFERENCE_NAME_APPLICATION_LOCALE,
MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();
    editor.putString(PREFERENCE_KEY_APPLICATION_LOCALE, localeCode);
    editor.commit();
}
```

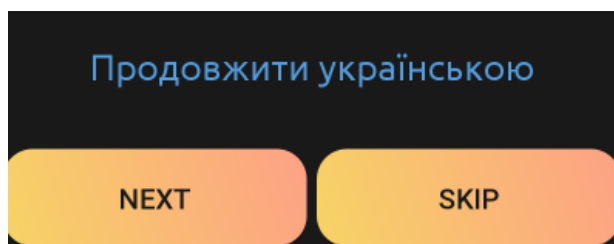


Рисунок 3.8 - Кнопка, що змінює мову застосунку

Вигляд форми на різних мовах наведено на рисунках 3.9 - 3.10.

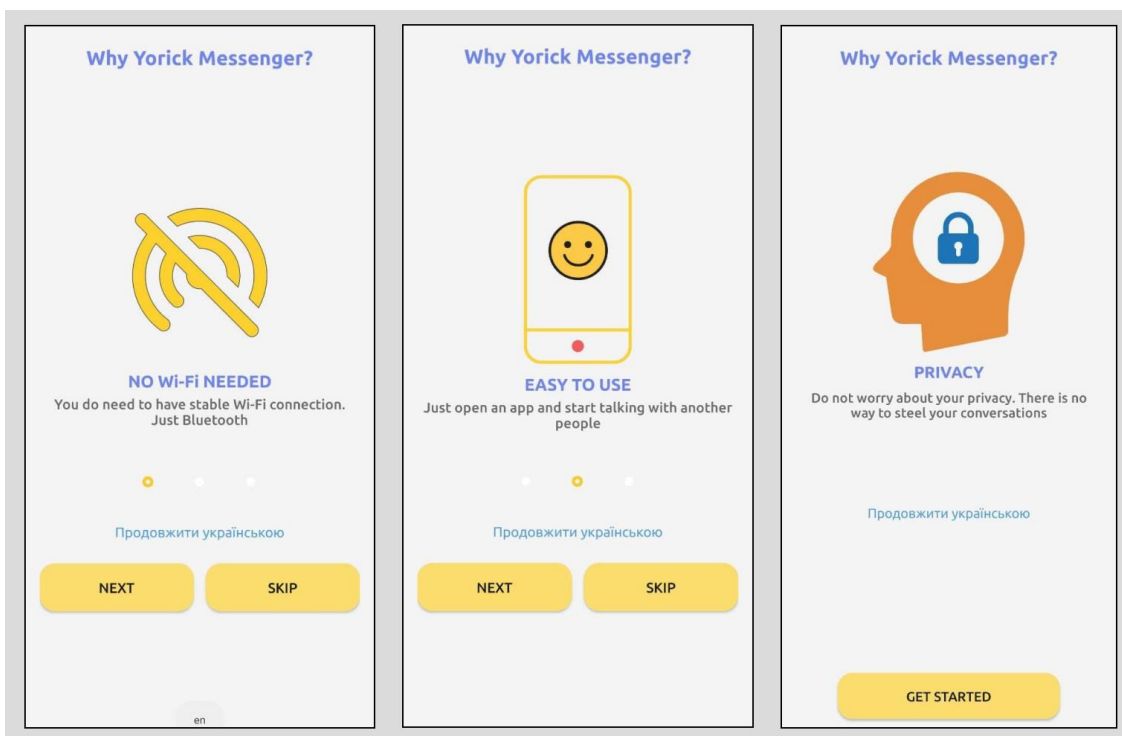


Рисунок 3.9 – Вигляд форми англійською мовою

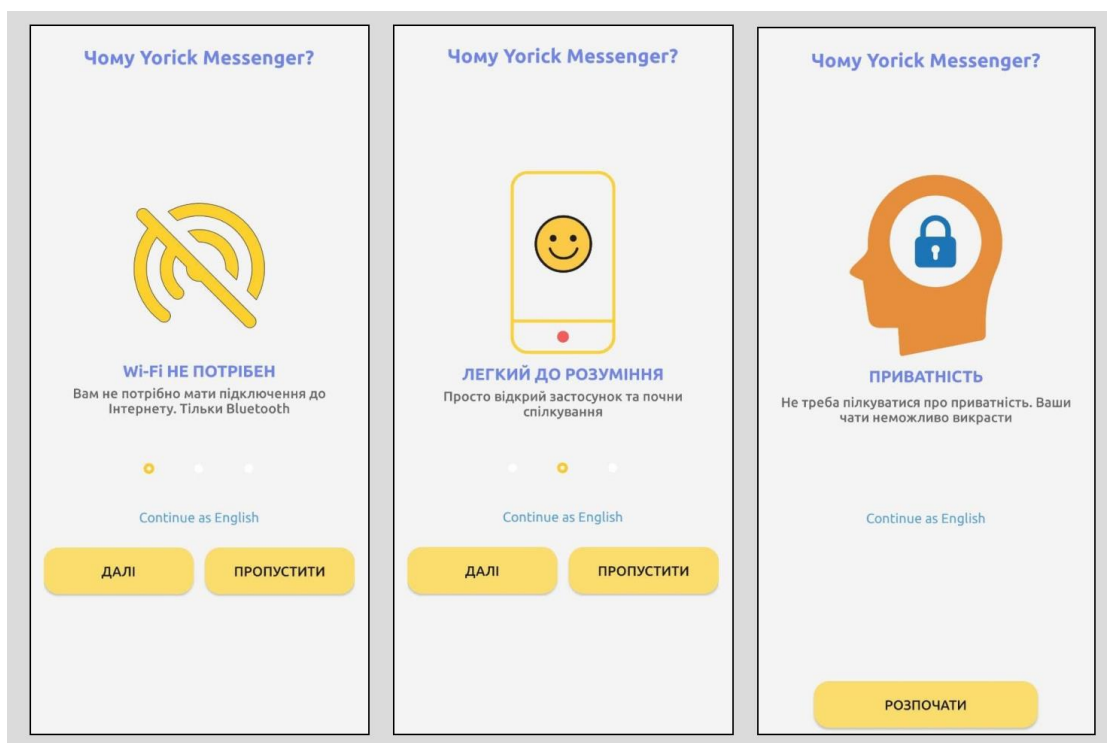


Рисунок 3.10 – Вигляд форми українською мовою

На рисунку 3.11 наведено діаграму класу (NewUserSupportActivity) для форми привітання:

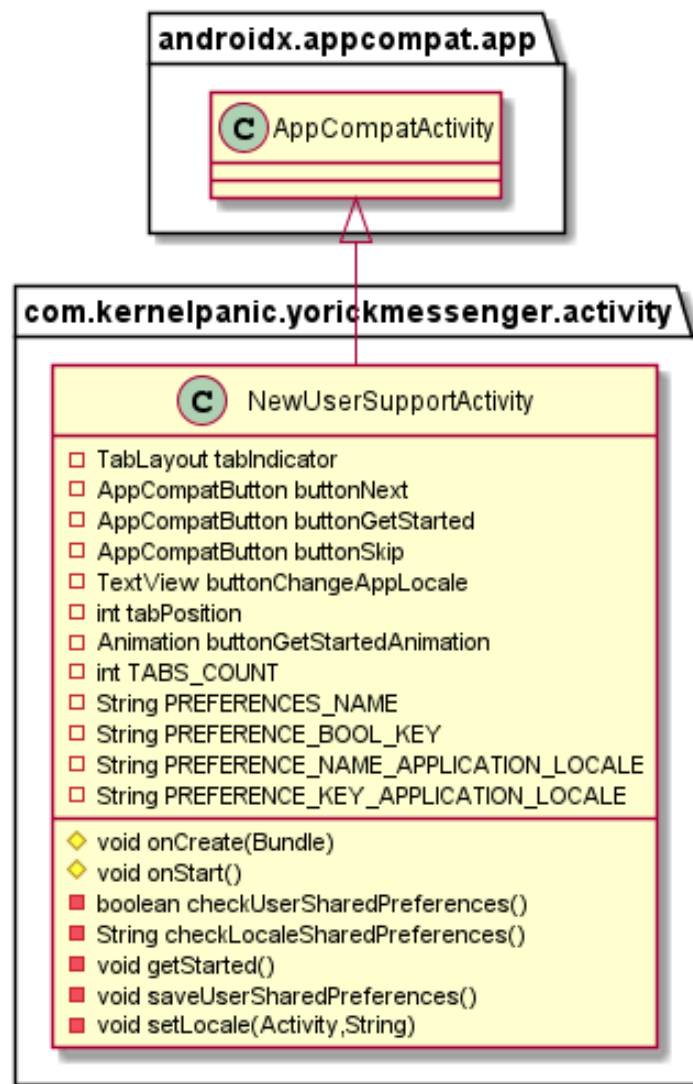


Рисунок 3.11 - Діаграма класів для форми привітання

3.5 Форма відрисовки головного користувацького інтерфейсу

Більшість форм застосунку є фрагментами, що надає застосунку швидкість при навігації, бо не потрібно постійно відкривати та закривати форми, можна просто змінити поточний фрагмент у відповідному контейнері. Структура головного “шару” застосунку містить верхню панель інструментів та контейнер для фрагментів (рис. 3.12).

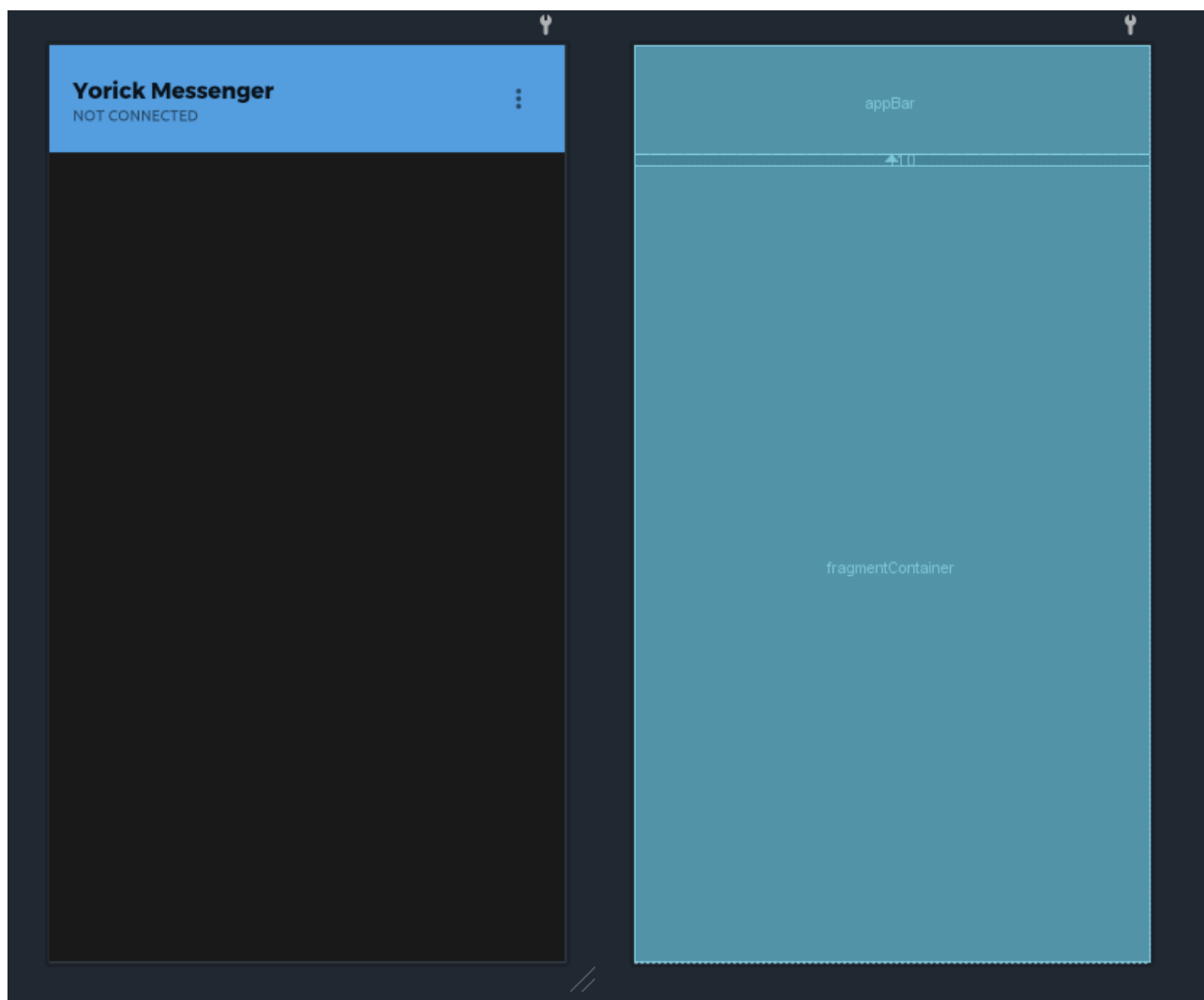


Рисунок 3.12 - Структура форми

Також є бокове меню, яке можна відкрити свайпом, або натиснувши на “menu hamburger” у лівій частині панелі інструментів (рис. 3.13).

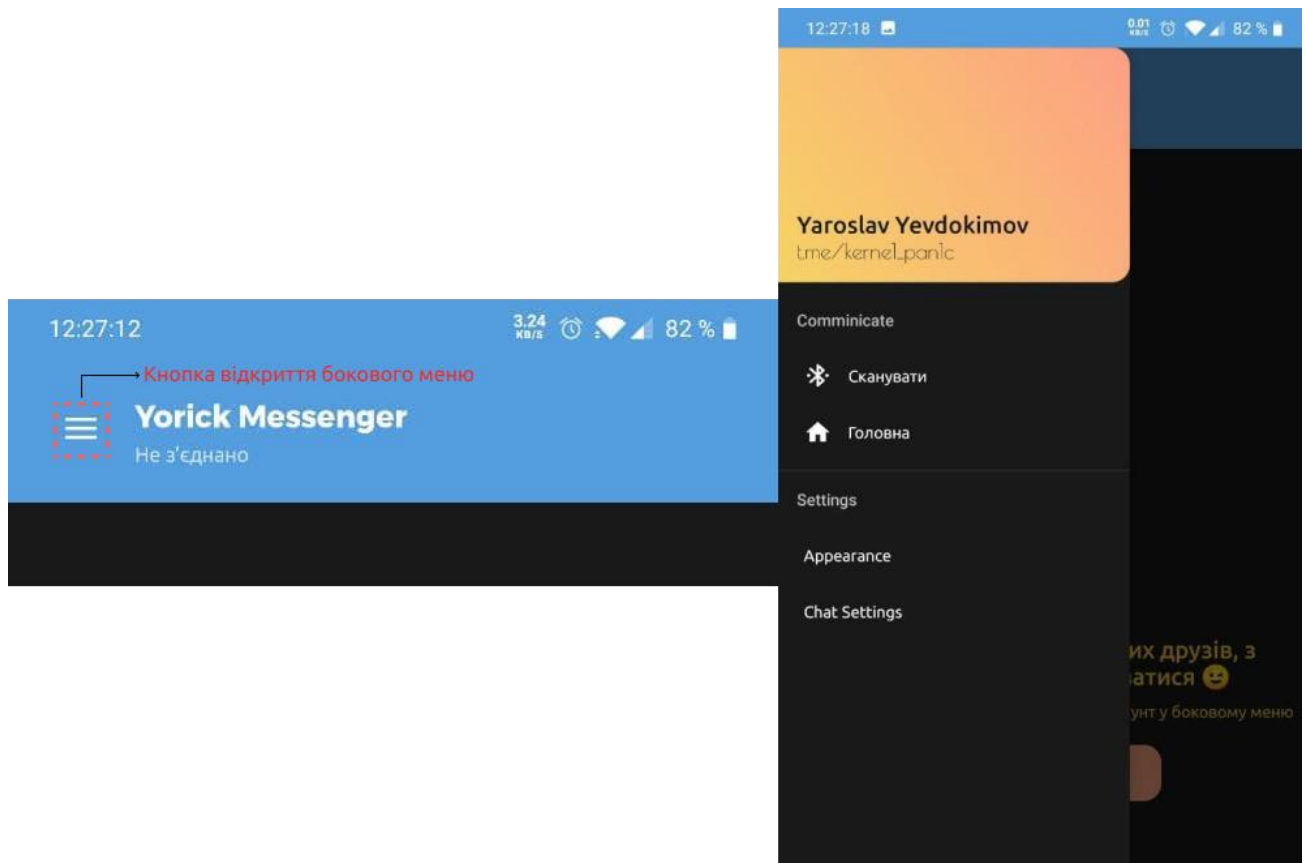


Рисунок 3.13 - Бокове меню

Головна форма також за допомогою функцій, що надає нам `SharedPreferences` перевіряє, чи є у користувача профіль. У випадку, якщо його немає, то відкривається фрагмент створення профілю:

```
fragmentManager = getSupportFragmentManager();
fragmentManager.beginTransaction()
    .setReorderingAllowed(true)
    .add(R.id.fragmentContainer, CreateProfileFragment.class, null)
    .commit();
```

На рисунку 3.14 наведено діаграму класу (`MainAppActivity`) для головної форми:

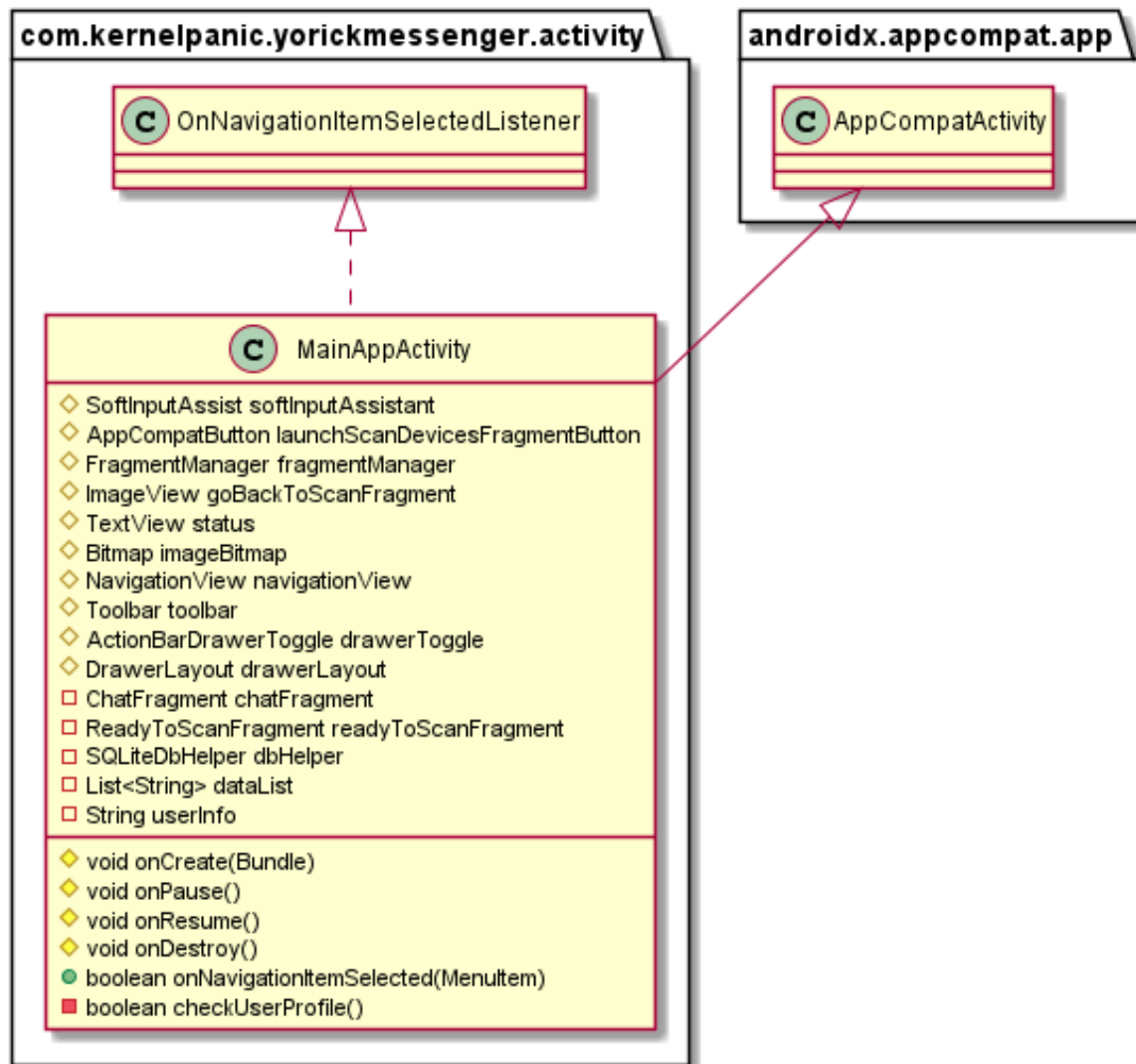


Рисунок 3.14 - Діаграма класів для головної форми

3.6 База даних

3.6.1 Чому SQLite?

Виходячи з того, що застосунок не має та не повинен мати доступ до Інтернету база даних є локальною на пристрої користувача. База зберігає дані про ім'я користувача, а також повинна зберігати відправлені та отримані повідомлення.

Тому був вибір: використовувати та зберігати в пам'яті користувацького пристрою JSON-файли формату. Але це породило би труднощі при зберіганні повідомлень, бо один такий запис має багато параметрів, а їх може бути

декілька. Але зручно підходить для зберігання даних про користувача, бо один запис містить тільки дані про ім'я.

Для простоти використання, розробки та підтримку було вирішено зробити бази за допомогою SQLite для користувацьких даних, та для даних про повідомлення.

3.6.2 Клас-помічник SQLiteOpenHelper

Для спрощення роботи з базою було створено клас, який виконує необхідні операції: створення таблиць, оновлення бази, додавання записів, робота з SQL запитамі. При створенні бази до методу onCreate() передається як параметри назва бази та її версія. За допомогою версії та методу onUpgrade() можна оновлювати базу [15], [16]:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_MESSAGES);
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_USER_INFO);
    onCreate(db);
}
```

Створення таблиць відбувається за допомогою виконання стандартних команд SQL:

```
//Executing a SQL command to CREATE messages table
db.execSQL("CREATE TABLE " + TABLE_MESSAGES + "(" +
    DEVICE_MAC_ADDRESS + " VARCHAR, " +
    CONTENTS + " VARCHAR, " +
    USER_NAME + " VARCHAR, " +
    TYPE + " INTEGER, " +
    TIMESTAMP + " TIMESTAMP" + ")");

//Executing a SQL command to CREATE user's info table
db.execSQL("create table " + TABLE_USER_INFO + "(" +
```

```
uID + " integer primary key autoincrement, " + FULLNAME + " text" +
")");
```

Додавання запису до таблиці відбувається за допомогою класу ContentValues та вбудованого методу insert(). Для спрощення роботи зі змінними було створено модель User, яка містить у собі дані для користувача (рис. 3.15). Приклад додавання рядку:

```
SQLiteDatabase db = getWritableDatabase();

ContentValues values = new ContentValues();
values.put(uID, user.getId());
values.put(FULLNAME, user.getFullname());

db.insert(TABLE_USER_INFO, null, values);
db.close();
```

```
package com.kernelpanic.yorickmessenger.database;

public class User {
    public int id;
    public String name;
    public String fullname;

    public User(String fullname) { this.fullname = fullname; }

    //Empty constructor
    public User() {
    }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getFullname() { return fullname; }
}
```

Рисунок 3.15 - Модель “Користувач”

Окрім моделі “Користувач” також розроблена модель “Повідомлення”. По аналогії з моделлю “Користувач” містить дані про структуру повідомлення (рис. 3.16).

```
public class ChatMessage {

    private String macAddress;
    private int type;
    private String content;
    private String username;
    private Long currentTime;

    // Empty constructor
    public ChatMessage() {
    }

    public ChatMessage(String macAddress, int type, String content, String username, Long currentTime) {
        this.macAddress = macAddress;
        this.type = type;
        this.content = content;
        this.username = username;
        this.currentTime = currentTime;
    }

    public String getMacAddress() { return macAddress; }

    public void setMacAddress(String macAddress) { this.macAddress = macAddress; }

    public int getType() { return type; }

    public void setType(int type) { this.type = type; }

    public String getContent() { return content; }

    public void setContent(String content) { this.content = content; }

    public String getUsername() { return username; }

    public void setUsername(String username) { this.username = username; }

    public Long getCurrentTime() { return currentTime; }

    public void setCurrentTime(Long currentTime) { this.currentTime = currentTime; }

}
```

Рисунок 3.16 - Модель “Повідомлення”

Зчитування рядків з бази проводиться за допомогою вбудованих функцій інструменту Cursor та SQL.

Приклад використання лише інструменту Cursor:

```

SQLiteDatabase db = this.getReadableDatabase();

Cursor cursor = db.query(TABLE_USER_INFO, null, null, null, null, null, null);
if (cursor != null)
    cursor.moveToFirst();

String userInfo = cursor.getString(1);
cursor.close();
db.close();
return userInfo;

```

Приклад використання інструменту Cursor та SQL:

```

SQLiteDatabase db = getReadableDatabase();
Toast.makeText(context, DEVICE_MAC_ADDRESS, Toast.LENGTH_SHORT).show();
return db.rawQuery("SELECT * FROM messages WHERE " + DEVICE_MAC_ADDRESS + "=?",
    new String[]{deviceMac});

```

На рисунку 3.17 наведено діаграму класів для класу SQLiteDatabaseHelper.

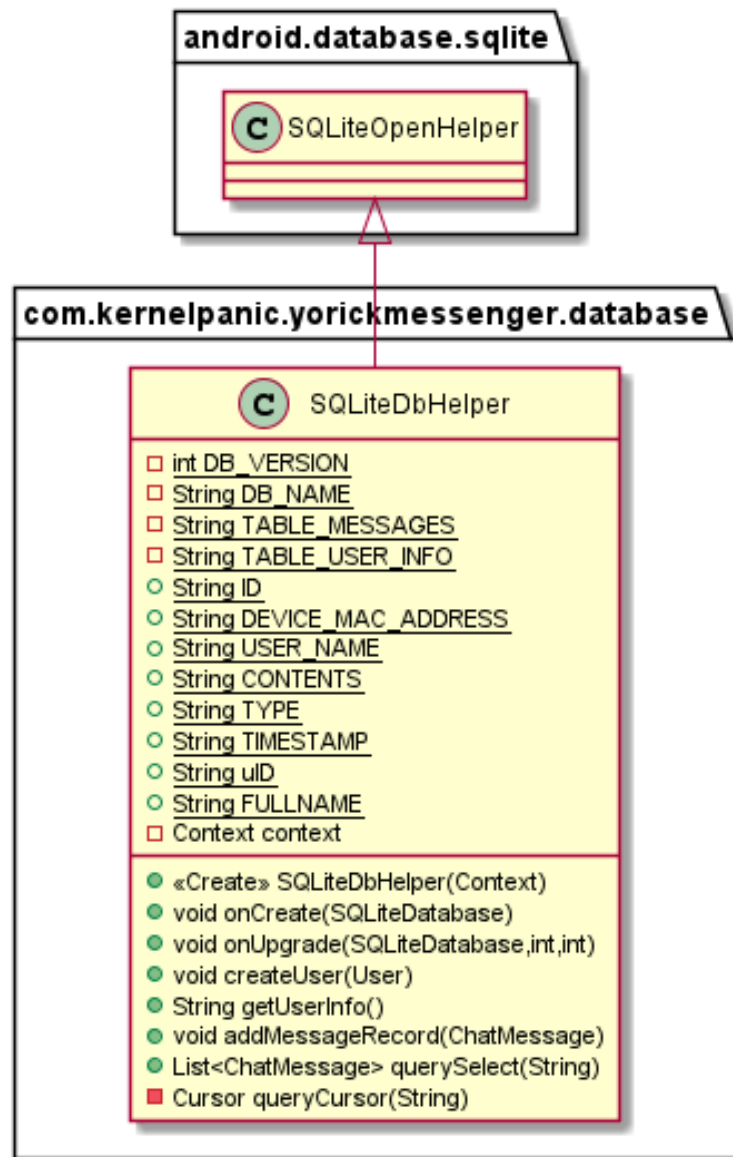


Рисунок 3.17 - Діаграма класів для класу-помічника

3.6.3 Використання бази

3.6.3.1 Створення користувацького профілю. Налаштування профіля: зчитування бази

Фрагмент, що викликається при першому запуску, коли у користувача ще немає профілю працює з функціями класу-помічника та додає до бази рядок з ім'ям користувача:

```

sqlHelper = new SQLiteDatabaseHelper(getActivity().getApplicationContext());
SQLiteDatabase db = sqlHelper.getWritableDatabase();

createProfileButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String fullname = nameField.getText().toString();
        readyToScanFragment = new ReadyToScanFragment();

        User user = new User(fullname);
        sqlHelper.createUser(user);
        Toast.makeText(getActivity(), "Added new user",
Toast.LENGTH_SHORT).show();

        SharedPreferences sharedPreferences =
getActivity().getSharedPreferences(PREFERENCE_NAME_USER_PROFILE,
Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putBoolean(PREFERENCE_KEY_USER_PROFILE, true);
        editor.commit();

        getActivity().finish();
    }
}

```

Головна форма теж має доступ до бази, вона ініціалізує профіль користувача: зчитує ім'я з бази та додає його до заголовку бокового меню:

```

dbHelper = new SQLiteDatabaseHelper(this);
navigationView = findViewById(R.id.navigationView);
navigationView.setNavigationItemSelectedListener(this);

View navHeaderView = navigationView.getHeaderView(0);
TextView userLabel = navHeaderView.findViewById(R.id.userInfo);
userLabel.setText(userInfo);

```

3.6.3.2 Фрагмент чату: обмін користувацькими даними, додавання повідомлень до бази

Фрагмент чату використовує бази для зчитування ім'я користувача та обміну цими даними з іншим пристроєм, коли вони з'єднуються. Для цього було створено метод, який за допомогою об'єкту `dbHelper` зчитує ім'я, а потім за допомогою класу `BluetoothChatService` відправляє до іншого пристрою, коли стан змінюється на "ПІДКЛЮЧЕНО" (`STATE_CONNECTED`):

Метод обміну:

```
private void onConnectDataExchange() {
    databaseUsername = dbHelper.getUserInfo();
    chatService.sendData(databaseUsername);
}
```

Виклик методу при зміні стану:

```
case BluetoothChatService.STATE_CONNECTED:
    try {
        (...)
        onConnectDataExchange();
    }
    (...)
}
```

Для додавання до бази запису з повідомленням використовується модель "Повідомлення". До об'єкту передаються наступні параметри: MAC-адреса з'єданого пристрою, тип повідомлення (відправлено або прочитано), безпосередньо повідомлення, ім'я користувача (відправника або отримувача), час відправки:

```
dbHelper.addMessageRecord(new ChatMessage(connectedDeviceBluetoothAddress,
Constants.MESSAGE_TYPE_RECEIVED, readMessage,
receivedUsername, currentTimer));
```

3.7 Допоміжні класи

3.7.1 Клас SoftInputAssist

При розробці застосунку виникла проблема при спробах “налагодити” клавіатуру так, щоб при натискання на поле вводу, клавіатура не закривала собою місце, а “додавалась” до існуючої структури форми. Для цього було взято клас SoftInputAssist [17].

3.7.2 Інтерфейс системних констант

Для спрощеного доступу до часто використовуваних змінних було створено інтерфейс Constants, що надає застосунку доступ до необхідних змінних з будь-якого класу. Зміст класу буде наведено на рисунку 3. за допомогою діаграми класів.

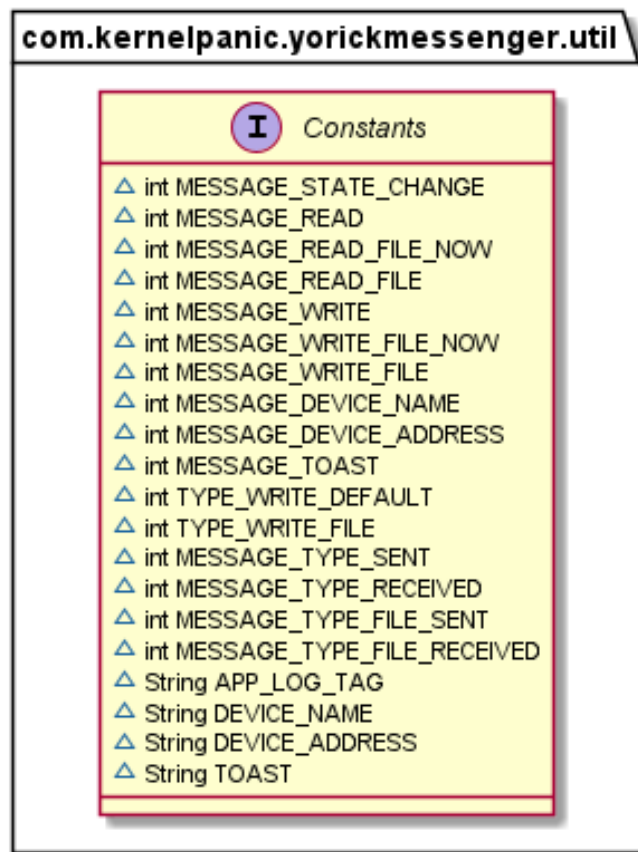


Рисунок 3.18 - Зміст інтерфейсу констант

4 ЕКСПЛУАТАЦІЯ ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Діаграма послідовностей

Для зрозуміння життєвого цикла затосунку було розроблено діаграму послідовностей, яка демонструє як та коли починають і/або закінчують роботу деякі компоненти затосунку.

Діаграму наведено на рисунку 4.1.

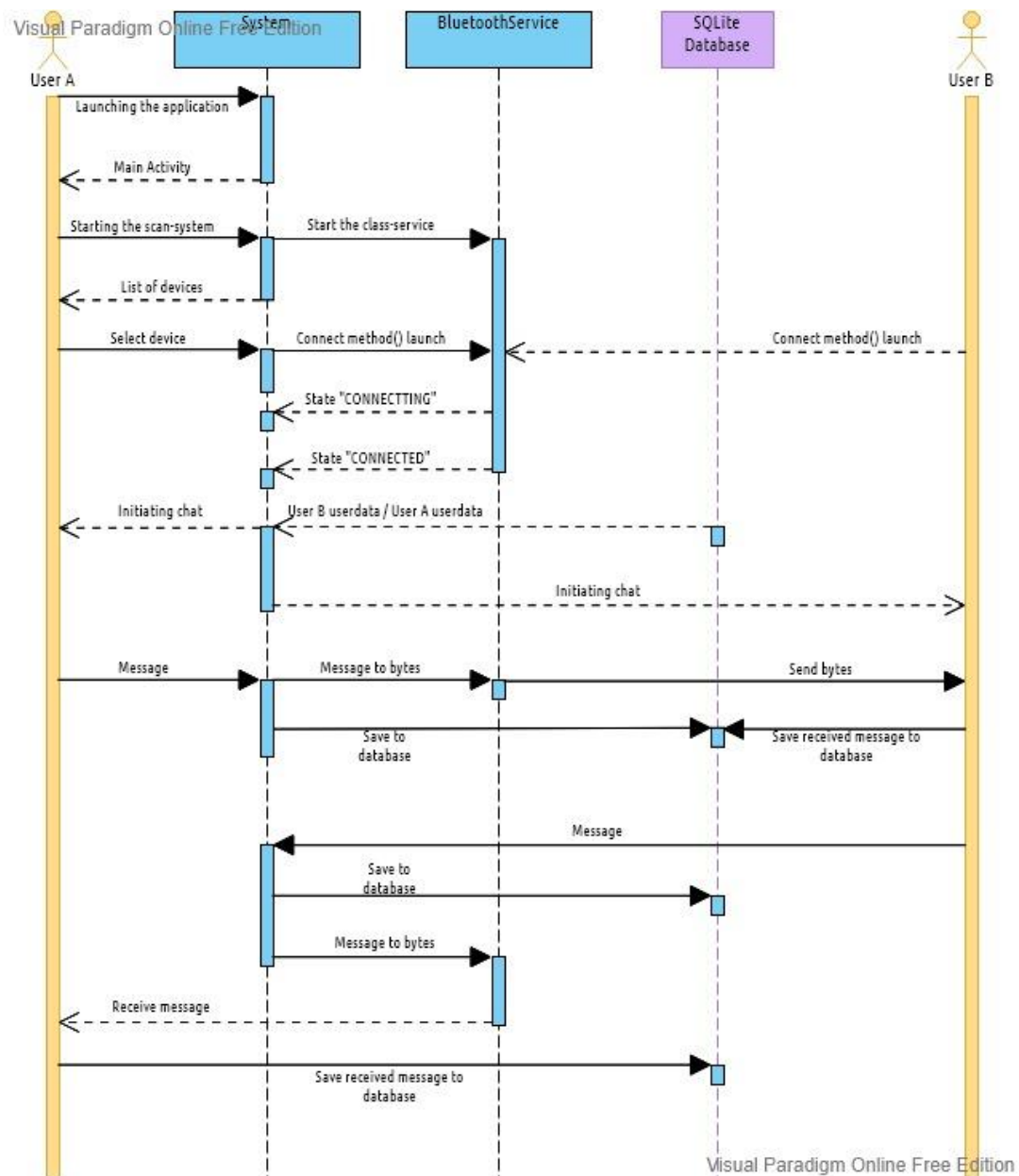


Рис 4.1 – Діаграма послідовностей

Опис діаграми послідовностей.

Крок 1. Користувач запускає застосунок, звертаючись до системи його пристрою.

Крок 2. Система користувача запускає застосунок та повертає вікно - головне вікно застосунку (MainAppActivity).

Крок 3. Користувач запускає форму сканування, перед цим система починає готувати фрагмент чату для роботи: починає роботу клас-сервіс та очікує. Система повертає користувача список знайдених пристроїв.

Крок 4. Користувач обирає пристрій із списку – система викликає метод класа-сервісу для виконання підключення на основі того чи є пристрій клієнтом або сервером. Якщо клієнтом, то виконується підключення до відкритого серверу. Якщо сервером, то виконується просліховування та приймання вхідного підключення.

Крок 5. Після вдалого з'єднання клас-сервіс повертає стан «ПІДКЛЮЧЕНО» до обоїх користувачів.

Крок 6. За допомогою бази даних достається інформація про ім'я профіля та користувачі обмінюються цією інформацією.

Крок 7. Чат готов до роботи. Користувачі мають змогу відправляти та приймати повідомлення один від одного: введене повідомлення передається до системи, система конвертує його до пакету байтів та відправляє до сервісу, який вже виконує відправку до іншого пристрою, або приймання від іншого пристрою. Також система записує до база даних відправлене та зчитане повідомлення.

4.2 Тестування застосунку

Тестування застосунку проводилось за допомогою декількох перевірок на різних пристроях з різними версіями Android та оболонками.

Список проведених перевірок:

- перевірка адаптивності інтерфеса застосунку;
- перевірка стабільності роботи;
- перевірка завантаження застосунку;
- перевірка використання батареї;
- перевірка на доступ до технологій Bluetooth (підключення, сканування, робота з повідомленнями);
- перевірка на роботу в фоні.

Список пристроїв, на яких проводилось тестування наведено у таблиці 4.1.

Таблиця 4.1 – Список пристроїв та результати тестування

№	Виробник	Модель	Версія Android	Оболонка	Результати перевірок
1	Xiaomi	Mi A1	10	crDroid (Custom Unofficial)	Проблеми з адаптивністю інтерфейсу
2	Xiaomi	Mi Note 10 Lite	11	MIUI	Критичні проблеми при роботі з файловою системою
3	OnePlus	6T	10	Oxygen OS	Незначні та нестабільні помилки при роботі з Bluetooth. Допомогає перезапуск пристрою
4	Huawei	MediaPad T3 10	7	EMUI	Проблеми за адаптивністю – планшет

На пристрої під номером 2 було виявлено критичні проблеми при роботі з файловою системою – на 11 версії Android було змінено права та методи доступу до файлової системи пристрою. Тому використання застарілих методів та методів обходу цієї системи не працюють.

На пристрої під номером 3 було виявлено нестабільну появу помилок в роботі сервісу та відмову виконувати підключення, визначити природу появи цих помилок не вдалося. Допомогає перезавантаження пристрою.

На пристрої під номером 1 в результаті невеликого екрану на деяких формах виникає проблема адаптивності.

ВИСНОВКИ

Під час роботи над проектом було ознайомлено з принципами та основами сучасного проектування мобільних застосунків за допомогою мови програмування Java. Розширено існуючі знання та навички написання програмного коду.

Для розробленого застосунку було створено репозиторій на GitHub: <https://github.com/katharosMelancholin/yorick-messenger-android>

Головним було ознайомитися та опанувати функції роботи з технологією Bluetooth, що й було виконано та продемонстровано у проекті під назвою “Yorick Messenger”.

Проектування застосунків під операційною системою Android має як переваги, так і свої недоліки. Головний недолік операційної системи Android, як не парадоксально це може прозвучати, – постійна зміна та оновлення. Так, застосунок, що був спроектован під застарілі версії Android може навіть не імпортуватися до середовища розробки Android Studio. З однією такою проблемою зустрівся і я, намагаючись реалізувати функції обміну не тільки повідомлення, але й файлами – на версії Android під кодовим ім’я “R” (11) було остаточно змінено правила та методи доступу до файлової системи пристрою. Тепер, застосунок не має змоги просто зробити директорію або файл у пам’яті, застосунок має доступ тільки до приватної app-specific директорії. Тому, для того, щоб проводити маніпуляції з файлом його необхідно спочатку помістити до цієї директорії, а потім вже копіювати куди потрібно. Складності виникли з часом та з посередньою документацією, що тільки більше збивала з пантелику.

Застосунок адаптовано під два види тем: світлу та темну.

Також було проведено декілько перевірок на різних пристроях. Перевірки ще раз довели, як на стабільність роботи застосунку впливають різні оболонки та версії Android.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. SMS [Електрон. ресурс] – Режим доступу: <https://ru.wikipedia.org/wiki/SMS>
2. Приложение Bluetooth Chat: [Електрон. ресурс]. - Режим доступу: <https://play.google.com/store/apps/details?id=com.glodanif.bluetoothchat>
3. Приложение Bluetooth Chat [Електрон. ресурс]. - Режим доступу: <https://github.com/NYIT-CSCI455-2015-Spring/BluetoothChat>
4. Приложение BlueChat [Електрон. ресурс]. - Режим доступу: <https://github.com/webianks/BluetoothChat>
5. Документація Android Source Project [Електрон. Ресурс]. - Режим доступу: <https://source.android.com/setup/start/build-numbers>
6. Гриффитс Д. Head First. Программирование для. Android 2-е изд. / Д. Гриффитс, Д. Гриффитс., 2018. – 912 с.
7. Olsson M. C# 8 Quick Syntax Reference: A Pocket Guide to the Language, APIs, and Library / Mikael Olsson., 2020.
8. Android [Електрон. ресурс] – Режим доступу: <https://ru.wikipedia.org/wiki/Android>.
9. Skeen J. Kotlin Programming: The Big Nerd Ranch Guide / J. Skeen, D. Greenhalgh., 2018.
10. Flutter Dev. Docs [Електрон. ресурс] – Режим доступу: <https://flutter.dev/docs>.
11. Android Studio [Електрон. ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/Android_Studio.
12. Linear Layout | Android Developers [Електрон. ресурс] – Режим доступу: <https://developer.android.com/guide/topics/ui/layout/linear>.
13. Relative Layout | Android Developers [Електрон. ресурс] – Режим доступу: <https://developer.android.com/guide/topics/ui/layout/relative>.

14. Constraint Layout | Android Developers [Электрон. ресурс] – Режим доступа: <https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout>.

15. Forta B. SQL in 10 Minutes, Sams Teach Yourself / Ben Forta., 2020.

16. Урок 34. Хранение данных. SQLite [Электрон. ресурс] – Режим доступа: <https://startandroid.ru/ru/uroki/vse-uroki-spiskom/74-urok-34-hranenie-dannyh-sqlite.html>.

17. grennis SoftInputAssist [Электрон. ресурс] – Режим доступа: <https://gist.github.com/grennis/2e3cd5f7a9238c59861015ce0a7c5584>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

Найменування продукту – Android-застосунок «Yorick Messenger». Застосунок «YorickMessenger» розроблено та призначено для організації спілкування на смартфонах за допомогою використання технології Bluetooth.

A.2 Підстава для розробки

Підставою для розробки служить наказ 30 березня 2021р. №103 Національного Університету «Запорізька Політехніка» на дипломну кваліфікаційну роботу на тему «Мобільний додаток для спілкування за допомогою технології Bluetooth. Mobile Application for Communication via Bluetooth».

A.3 Призначення розробки

Мета створення розробки програмного продукту, що дозволяє користувачам на пристроях яких встановлено застосунок виконувати спілкування за допомогою технології Bluetooth; виконувати підключення, відправку файлів та звичайних повідомлень.

A.4 Вимоги до програми чи програмного виробу

A.4.1 Вимоги до функціональних характеристик

Функції, які має виконувати застосунок:

- сканувати мережу Bluetooth, знаходити нови пристрої;
- виконувати підключення до іншого віддаленого пристрою Bluetooth;
- надавати користувач можливість спілкуватися: відправляти повідомлення та файли.

Вхідними даними можна вважати ім'я користувача та MAC-адреса пристрою.

Вихідними даними є різноманітні системні повідомлення та повідомлення, отримані від іншого пристрою.

A.4.2 Вимоги до надійності

Головна вимога з надійності до застосунку – стабільна робота в будь-яких мовах, без доступу до мережі інтернет. Необхідно запобігти закриття застосунку з причини помилки, запобігти невірні дії користувача.

A.4.3 Умови експлуатації

Загальні умови експлуатації співпадають з умовами експлуатації пристрою, на якому відбувається функціонування застосунку. Для коректної роботи застосунку необхідно мати Bluetooth та Android версії 5 або вище.

A.4.4 Вимоги до складу й параметрів технічних засобів

Технічні вимоги до обладнання:

- android 5 або вище;
- 64-bit CPU з як мінімум 4x Arm cores
- 1.5 Гб або більше оперативної пам'яті;
- розмір вільної пам'яті пристрою не менше 8Мб;
- пристрій має Bluetooth.

A.4.5 Вимоги до інформаційної й програмної сумісності

Програмні вимоги:

- версія Android 5 або вище;

- доступ до Bluetooth;
- сервіс віртуальної клавіатури.

A.4.6 Вимоги до маркування й упакування

Застосунок може бути записаний на носій ємністю не менше 100Мб. Або завантажений до електронної крамниці застосунків Google. У цьому випадку застосунок повинен мати логотип «Yorick Messenger» та дані про розробника.

A.4.7 Вимоги до транспортування й збереження

Вимоги до транспортування і зберігання аналогічні вимогам, що пред'являються до гнучких і лазерних дисків.

A.5 Вимоги до програмної документації

Програмна документація повинна включати: пояснювальну записку на розроблені застосунки, вихідний код програми.

A.6 Техніко-економічні показники

Монетизацію не передбачено шляхом вбудовування реклами у застосунок, бо необхідно мати доступ до інтернету. Монетизація може бути реалізована в якості платного скачування продукту.

A.7 Порядок контролю й приймання

Передбачено захист дипломної кваліфікаційної роботи у встановлені керівником роботи строки.

ДОДАТОК Б
Текст програми

```

public class BluetoothChatService {

    /**
     * BluetoothChatService class made to describe the overall logic of
     * Bluetooth connections interaction
     *
     * Here we have the ConnectThread, ManageThread, ServerThread
     *
     * One of the main value here is UUID - this value is necessary to work
     * without that we could not start the BluetoothSocket
     */

    private static final String app_name = "YorickMessenger";
    private static final UUID MY_UUID = UUID.fromString("8ce255c0-200a-11e0-
ac64-0800200c9a66");

    private static final String TAG = "Y.Messenger-Logs";

    private final BluetoothAdapter bluetoothAdapter;
    private final Handler handler;
    // Thread for accepting incoming connections
    // Thread для приймання зв'язків, що надходять до пристрою
    private ServerThread serverThread;
    /**
    Thread for connecting to remote devices
    Thread для з'єднання з пристроєм
    */
    private ConnectThread connectThread;
    /**
    Thread that allows to manage with read and write methods
    Thread що дозволяє виконувати операції відправки та отримання
    */
    private ManageThread manageThread;
    private int mState;
    private ContentResolver contentResolver;

    private Context mAndroidContext;

    public static final int STATE_NONE = 0;
    public static final int STATE_LISTEN = 1;
    public static final int STATE_CONNECTING = 2;
    public static final int STATE_CONNECTED = 3;

    private static String FILE_PATH;

    private ExecutorService executor = Executors.newSingleThreadExecutor();

    /**
     * Just a constructor
     * Конструктор
     * @param context - androidContext
     * @param handler - object of the Handler class that defined in {@link
ChatFragment}
     *
     * об'єкт класу Handler, що описан у фрагменті чату
     */
    public BluetoothChatService(Context context, Handler handler) {
        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        mState = STATE_NONE;
        mAndroidContext = context;
        this.handler = handler;
        contentResolver = context.getContentResolver();
    }
}

```

```

        FILE_PATH = context.getExternalFilesDir(null).getAbsolutePath() +
        "/YorickCache/";
    }

    /**
     * This method created to change the state. Changing the state was made to
     listen to the
     * action that we performing right now. E.g: LISTEN, CONNECTING
     * Цей метод створений для зміни стану. Стан змінюється для того, щоб
     слідуватися за тим,
     * що ми робимо
     * @param state - value to what we change current state (mState); зміна, на
     яку ми змінємо стан
     */
    private synchronized void setState(int state) {
        Log.i(TAG, "The setState() method called, state " + mState + " is going
to -> " + state);
        mState = state;
        handler.obtainMessage(Constants.MESSAGE_STATE_CHANGE, state, -
1).sendToTarget();
    }

    /**
     * No need to describe
     * @return - state
     */
    public synchronized int getState() { return mState; }

    /**
     * Method that starts the ServerThread (whom we need to perform bluetooth
connections)
     * Also we need to cancel all previous connections and threads if they exist
     * Метод, що запускає ServerThread, який нам потрібен для підключення
     * Також нам необхідно відключити вже існуючі Thread
     */
    public synchronized void start() {
        Log.d(TAG, "Initiating the start procedure");
        if (connectThread != null) {
            connectThread.cancel();
            connectThread = null;
        }

        if (manageThread != null) {
            manageThread.cancel();
            manageThread = null;
        }

        setState(STATE_LISTEN);
        if (serverThread == null) {
            serverThread = new ServerThread();
            serverThread.start();
        }
    }

    /**
     * Method that perform connect to another remote bluetooth device
     * Метод, який виконує з'єднання до іншого пристрою
     * @param device - object of BluetoothDevice class, what we get from the
{@link ChatFragment}
     */
    public synchronized void connect(BluetoothDevice device) {

```

```

        Log.d(TAG, "Performing the attempt to connect to the remote bluetooth
device, MAC: "
            + device.getAddress() + " name: " + device.getName());

        if (mState == STATE_CONNECTING) {
            if (connectThread != null) {
                connectThread.cancel();
                connectThread = null;
            }
        }

        connectThread = new ConnectThread(device);
        connectThread.start();
        setState(STATE_CONNECTING);
    }

    /**
     * Method that starts the ManageThread
     * Метод, що запускає ManageThread
     * @param socket - object of the BluetoothSocket class; об'єкт класу
BluetoothSocket
     * @param device - object of the BluetoothDevice class; об'єкт класу
BluetoothDevice
     */
    public synchronized void connected(BluetoothSocket socket, BluetoothDevice
device) {
        Log.i(TAG, "Connected! BluetoothDevice: " + device);
        if (connectThread != null) {
            connectThread.cancel();
            connectThread = null;
        }

        if (manageThread != null) {
            manageThread.cancel();
            manageThread = null;
        }

        if (serverThread != null) {
            serverThread.cancel();
            serverThread = null;
        }

        manageThread = new ManageThread(socket);
        manageThread.start();

        Message message = handler.obtainMessage(Constants.MESSAGE_DEVICE_NAME);
        Message messageAddress =
handler.obtainMessage(Constants.MESSAGE_DEVICE_ADDRESS);
        Bundle bundle = new Bundle();
        bundle.putString(Constants.DEVICE_NAME, device.getName());
        Bundle bundleAddress = new Bundle();
        bundleAddress.putString(Constants.DEVICE_ADDRESS, device.getAddress());
        message.setData(bundle);
        messageAddress.setData(bundleAddress);

        handler.sendMessage(message);
        handler.sendMessage(messageAddress);

        setState(STATE_CONNECTED);
    }

    /**
     * Method that stops all threads

```

```

    * Метод, що припиняє роботу сервісу
    */
    public synchronized void stop() {
        if (connectThread != null) {
            connectThread.cancel();
            connectThread = null;
        }

        if (manageThread != null) {
            manageThread.cancel();
            manageThread = null;
        }

        if (serverThread != null) {
            serverThread.cancel();
            serverThread = null;
        }

        setState(STATE_NONE);
    }

    public void sendData(String message) {
        ManageThread thread;
        synchronized (this) {
            if (mState != STATE_CONNECTED) {
                return;
            }
            thread = manageThread;
        }
        thread.write(message);
    }

    public void sendFile(String filePath) {
        ManageThread thread;
        synchronized (this) {
            if (mState != STATE_CONNECTED) {
                return;
            }
            thread = manageThread;
        }
        thread.writeFile(filePath);
    }

    /**
     * Connection failed handler, sends to the user an error during the
     connection
     * Метод, що повідомляє користувача про помилку під час підключення до
     пристрою
     */
    private void connectionFailed() {
        Message message = handler.obtainMessage(Constants.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString("toast", "Unable to connect device");
        message.setData(bundle);
        handler.sendMessage(message);

        BluetoothChatService.this.start();
    }

    /**
     * Connection lost handler, sends to the user an error during the connection
     * Метод, що повідомляє користувача про втрату зв'язку з пристроєм
     */

```

```

private void connectionLost() {
    Message msg = handler.obtainMessage(Constants.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString("toast", "Device connection was lost");
    msg.setData(bundle);
    handler.sendMessage(msg);

    // Start the service over to restart listening mode
    BluetoothChatService.this.start();
}

/**
 * ServerThread - class that describe the logic of accepting the incoming
connections
 * ServerThread - клас, який описує логіку з'єднання одного пристрою з іншим
 */
private class ServerThread extends Thread {

    private final BluetoothServerSocket serverSocket;

    public ServerThread() {
        BluetoothServerSocket tmp = null;

        try {
            tmp =
bluetoothAdapter.listenUsingInsecureRfcommWithServiceRecord(app_name, MY_UUID);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        serverSocket = tmp;
    }

    public void run() {
        setName("ServerThread");
        BluetoothSocket socket;
        while (mState != STATE_CONNECTED) {

            try {
                socket = serverSocket.accept();
            } catch (IOException e) {
                e.printStackTrace();
                break;
            }

            if (socket != null) {
                synchronized (BluetoothChatService.this) {
                    switch (mState) {
                        case STATE_LISTEN:
                        case STATE_CONNECTING:
                            connected(socket, socket.getRemoteDevice());
                            break;
                        case STATE_NONE:
                        case STATE_CONNECTED:
                            try {
                                socket.close();
                            } catch (IOException e) {
                                e.printStackTrace();
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
}

public void cancel() {
    try {
        serverSocket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

}

/**
 * ConnectThread - class that describe the logic of connecting to another
device
 * Mean, here we are <b>trying</b> to connect to another device, while the
 * {@param ServerThread} trying to accept that connection
 * ConnectThread - клас, який описує логіку з'єднання одного пристрою з
іншим пристроєм
 * Тобто, у цьому класі ми виконуємо спробу з'єднатися з пристроєм, коли
ServerThread намагається
 * прийняти це з'єднання
 */
private class ConnectThread extends Thread {

    private final BluetoothSocket socket;
    private final BluetoothDevice device;

    public ConnectThread(BluetoothDevice device) {
        this.device = device;
        BluetoothSocket tmpSocket = null;

        try {
            //tmpSocket = (BluetoothSocket)
device.getClass().getMethod("createRfcommSocket", new Class[]{ int.class
}).invoke(device, 1);
            tmpSocket =
device.createInsecureRfcommSocketToServiceRecord(MY_UUID);
        } catch (Exception e) {
            e.printStackTrace();
        }
        socket = tmpSocket;
    }

    public void run() {
        setName("ConnectThread");

        // Always cancel discovery because it will slow down a connection
        bluetoothAdapter.cancelDiscovery();

        // Make a connection to the BluetoothSocket
        try {
            socket.connect();
        } catch (IOException e) {
            try {
                socket.close();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
            connectionFailed();
            return;
        }
    }
}

```

```

    }

    // Reset the ConnectThread because we're done
    synchronized (BluetoothChatService.this) {
        connectThread = null;
    }

    // Start the connected thread
    connected(socket, device);
}

public void cancel() {
    try {
        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

}

/**
 * ManageThread - class that allows to read and write data
 * ManageThread - клас, який дозволяє зчитувати та відправляти дані
 */
private class ManageThread extends Thread {

    private final BluetoothSocket socket;
    private final InputStream inputStream;
    private final DataInputStream dataInputStream;
    private final DataOutputStream dataOutputStream;
    private final OutputStream outputStream;

    public ManageThread(BluetoothSocket socket) {
        this.socket = socket;
        inputStream = null;
        outputStream = null;

        try {
            inputStream = socket.getInputStream();
            outputStream = socket.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
        inputStream = inputStream;
        dataInputStream = new DataInputStream(inputStream);
        outputStream = outputStream;
        dataOutputStream = new DataOutputStream(outputStream);
    }

    public void run() {

        while (true) {
            try {
                switch (dataInputStream.readInt()) {
                    case Constants.TYPE_WRITE_DEFAULT:
                        String message = dataInputStream.readUTF();
                        sendMessageToUI(Constants.MESSAGE_READ, message);
                        break;
                    case Constants.TYPE_WRITE_FILE:

```

```

        File destinationDirectory = new
File(mAndroidContext.getExternalFilesDir(null).getAbsolutePath() +
"/YorickCache/");

        if (!destinationDirectory.exists())
            destinationDirectory.mkdirs();
        String filename = dataInputStream.readUTF();
        long fileSize = dataInputStream.readLong();
        sendMessageToUI(Constants.MESSAGE_READ_FILE_NOW,
fileSize);

        long size = 0;
        int r;
        byte[] bytes = new byte[4 * 1024];
        FileOutputStream out = new
FileOutputStream(FILE_PATH + filename);
        while ((r = inputStream.read(bytes)) != -1) {
            out.write(bytes, 0, r);
            size += r;
            if (size >= fileSize) {
                break;
            }
        }
        if (Build.VERSION.SDK_INT >= 29) {
            ContentValues values = new ContentValues();
            values.put(MediaStore.Images.Media.DISPLAY_NAME,
filename + ".jpg");
            values.put(MediaStore.Images.Media.MIME_TYPE,
"image/jpeg");

            values.put(MediaStore.Images.Media.RELATIVE_PATH, Environment.DIRECTORY_PICTURES
+ "/Yorick Pictures");

            Uri imageUri =
contentResolver.insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values);
            Bitmap bitmap =
MediaStore.Images.Media.getBitmap(contentResolver, imageUri);
            out = (FileOutputStream)
contentResolver.openOutputStream(imageUri);
            bitmap.compress(Bitmap.CompressFormat.JPEG, 90,
out);

        }
        sendMessageToUI(Constants.MESSAGE_READ_FILE,
filename);

        break;
    }
} catch (IOException ioEX) {
    ioEX.printStackTrace();
    Log.d(TAG, "Receiving data error, exception: " +
ioEX.getMessage());
    BluetoothChatService.this.start();
    break;
}
}

public void write(final String message) {
    executor.execute(new Runnable() {
        public void run() {
            try {
                dataOutputStream.writeInt(Constants.TYPE_WRITE_DEFAULT);
                dataOutputStream.writeUTF(message);
            } catch (Throwable ex) {
                Log.e(TAG, "We have caught an exception: " + ex);
            }
            sendMessageToUI(Constants.MESSAGE_WRITE, message);
        }
    });
}

```

```

        }
    });
}

public void writeFile(final String filePath) {
    executor.execute(new Runnable() {
        public void run() {
            try {
                sendMessageToUI(Constants.MESSAGE_WRITE_FILE_NOW,
"Sendind file + " + filePath);
                FileInputStream fileInputStream = new
FileInputStream(filePath);
                File file = new File(filePath);
                dataOutputStream.writeInt(Constants.TYPE_WRITE_FILE);
                dataOutputStream.writeUTF(file.getName());
                dataOutputStream.writeLong(file.length());
                int r;
                byte[] bytes = new byte[4 * 1024];
                while ((r = fileInputStream.read(bytes)) != -1) {
                    dataOutputStream.write(bytes, 0, r);
                }
                sendMessageToUI(Constants.MESSAGE_WRITE_FILE,
file.getName());
            } catch (Throwable ex) {
                Log.e(TAG, "We have caught the exception: ", ex);
            }
        }
    });
}

public void cancel() {
    try {
        socket.close();
    } catch (IOException e) {
        Log.e(TAG, "Exception", e);
    }
}

}

private void sendMessageToUI(int what, Object object) {
    Message message = handler.obtainMessage();
    message.what = what;
    message.obj = object;
    handler.sendMessage(message);
}
}

```

ChatFragment.java

```

public class ChatFragment extends Fragment {

    private String sentUsername = null;
    private String receivedUsername = null;
    private String databaseUsername;

    private StringBuffer outStringBuffer;
    private String connectedDeviceBluetoothName = null;
    private String connectedDeviceBluetoothAddress =
"";
    private SQLiteDatabaseHelper dbHelper;
    private NotificationManager notificationManager;

```

```

        private      ArrayList<com.kernelpanic.yorickmessenger.util.Message>
messageList;

        private final String          TAG = "Y.Messenger-Logs";

        private final int              NOTIFY_ID = 1;
        private final String          CHANNEL_ID =
"YorickMessenger_Notify_Channel";

        private boolean isOnConnectExchangeDone = false;
        private boolean isImageWrite = false;

        private Handler handler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                super.handleMessage(msg);
                switch (msg.what) {
                    case Constants.MESSAGE_STATE_CHANGE:
                        switch (msg.arg1) {
                            case BluetoothChatService.STATE_CONNECTED:
                                try {
                                    btnAttach.setEnabled(false);
                                    btnSend.setEnabled(false);
                                    inputField.setEnabled(false);
                                    Snackbar.make(getView(), "Wait until we done
with transferring the user data", Snackbar.LENGTH_SHORT).show();
                                    toolbar =
getActivity().findViewById(R.id.appBar);

                                    toolbar.setSubtitle(getString(R.string.app_mainActivity_connected,
connectedDeviceBluetoothName));

                                    onConnectDataExchange();
                                    handler.postDelayed(new Runnable() {
                                        @Override
                                        public void run() {
                                            isOnConnectExchangeDone = true;
                                            btnSend.setEnabled(true);
                                            btnAttach.setEnabled(true);
                                            inputField.setEnabled(true);
                                            Snackbar.make(getView(), "Done",
Snackbar.LENGTH_SHORT).show();

                                            Toast.makeText(getActivity(),
"globalDeviceAddress: " + connectedDeviceBluetoothAddress,
Toast.LENGTH_SHORT).show();

                                            List<ChatMessage> chatList =
dbHelper.querySelect(connectedDeviceBluetoothAddress);
                                            for (ChatMessage chatMessage : chatList)
                                                messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(chatMessage.getContent(),
chatMessage.getCurrentTime(),

chatMessage.getType(), chatMessage.getUsername(), false));

                                            chatListAdapter.notifyDataSetChanged();

                                            chatView.smoothScrollToPosition(messageList.size());
                                        }
                                    }, 1500);
                                } catch (IllegalStateException ex) {
                                    Log.d(TAG, ex.getMessage());
                                }
                            }
                        }
                    }
                }
            }
        }

```

```

        notificationManager = (NotificationManager)
getActivity()
        .getApplicationContext()

        .getSystemService(Context.NOTIFICATION_SERVICE);
        Intent intent = new
Intent(getActivity().getApplicationContext(), MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK |
Intent.FLAG_ACTIVITY_NEW_TASK);
        PendingIntent pendingIntent =
PendingIntent.getActivity(getActivity().getApplicationContext(), 0, intent,
        PendingIntent.FLAG_UPDATE_CURRENT);
        NotificationCompat.Builder notificationBuilder =
        new
NotificationCompat.Builder(getActivity().getApplicationContext(), CHANNEL_ID)
        .setAutoCancel(false)
        .setWhen(System.currentTimeMillis())
        .setSmallIcon(R.drawable.ic_bluetooth_scan)
        .setContentIntent(pendingIntent)

        .setContentTitle(getString(R.string.app_name))

        .setContentText(getString(R.string.app_mainActivity_connected,
connectedDeviceBluetoothName))

        .setPriority(PRIORITY_HIGH)
        .setCategory(CATEGORY_MESSAGE)
        .setOnlyAlertOnce(true);
        createChannel(notificationManager);
        notificationManager.notify(NOTIFY_ID,
notificationBuilder.build());
        break;
        case BluetoothChatService.STATE_CONNECTING:
            Toast.makeText(getActivity(), "Connecting",
Toast.LENGTH_LONG).show();
            toolbar = getActivity().findViewById(R.id.appBar);

            toolbar.setSubtitle(getString(R.string.app_mainActivity_connecting));
            break;
        case BluetoothChatService.STATE_LISTEN:
        case BluetoothChatService.STATE_NONE:
            Toast.makeText(getActivity(), "Not connected",
Toast.LENGTH_LONG).show();
            toolbar = getActivity().findViewById(R.id.appBar);

            toolbar.setSubtitle(getString(R.string.app_mainActivity_notConnected));
            notificationManager = (NotificationManager)
getActivity()
        .getApplicationContext()

        .getSystemService(Context.NOTIFICATION_SERVICE);
        intent = new
Intent(getActivity().getApplicationContext(), MainActivity.class);
        intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK |
Intent.FLAG_ACTIVITY_NEW_TASK);
        pendingIntent =
PendingIntent.getActivity(getActivity().getApplicationContext(), 0, intent,
        PendingIntent.FLAG_UPDATE_CURRENT);
        Intent scanListIntent = new
Intent(getActivity().getApplicationContext(), ScanListActivity.class);
        PendingIntent pendingScanListIntent =
PendingIntent.getActivity(getActivity().getApplicationContext(), 0,
        scanListIntent,
PendingIntent.FLAG_ONE_SHOT);

```

```

        notificationBuilder =
            new
NotificationCompat.Builder(getActivity().getApplicationContext(), CHANNEL_ID)
                .setAutoCancel(false)
                .setWhen(System.currentTimeMillis())
                .setContentIntent(pendingIntent)

        .setSmallIcon(R.drawable.ic_bluetooth_scan)

        .setContentTitle(getString(R.string.app_name))
                .setContentText("Ready to connect")
                .setPriority(PRIORITY_HIGH)
                .setCategory(CATEGORY_MESSAGE)
                .setOnlyAlertOnce(true)

        .addAction(R.drawable.ic_bluetooth_scan, "SCAN", pendingScanListIntent);
        createChannel(notificationManager);
        notificationManager.notify(NOTIFY_ID,
notificationBuilder.build());
        break;
    }
    break;
    case Constants.MESSAGE_READ:
        Toast.makeText(getActivity(), "default read",
Toast.LENGTH_SHORT).show();
        String readMessage = (String) msg.obj;
        if (isOnConnectExchangeDone) {
            Long currentTimeR = System.currentTimeMillis();
            messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(readMessage,
                currentTimeR, Constants.MESSAGE_TYPE_RECEIVED,
receivedUsername, false));
            chatListAdapter.notifyDataSetChanged();
            chatView.smoothScrollToPosition(messageList.size());
            Toast.makeText(getActivity(), receivedUsername + "
received user", Toast.LENGTH_SHORT).show();
            dbHelper.addMessageRecord(new
ChatMessage(connectedDeviceBluetoothAddress, Constants.MESSAGE_TYPE_RECEIVED,
readMessage,
                receivedUsername, currentTimeR));
            break;
        } else {
            receivedUsername = (String) msg.obj;
            break;
        }
        // dbHelper.addMess(device_id, sent/received, devicename,
messgae);
    case Constants.MESSAGE_WRITE:
        Toast.makeText(getActivity(), "default write",
Toast.LENGTH_SHORT).show();
        Long currentTimeMillisW = System.currentTimeMillis();
        String writeMessage = (String) msg.obj;
        if (isOnConnectExchangeDone) {
            Toast.makeText(getActivity(), sentUsername + " sent
user", Toast.LENGTH_SHORT).show();
            Long currentTime = System.currentTimeMillis();
            messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(writeMessage,
                currentTimeMillisW,
Constants.MESSAGE_TYPE_SENT, sentUsername, false));
            chatListAdapter.notifyDataSetChanged();
            chatView.smoothScrollToPosition(messageList.size());

```

```

        dbHelper.addMessageRecord(new
ChatMessage (connectedDeviceBluetoothAddress, Constants.MESSAGE_TYPE_SENT,
writeMessage,

                sentUsername, currentTime));
        break;
    } else {
        sentUsername = (String) msg.obj;
        break;
    }
    case Constants.MESSAGE_READ_FILE_NOW:
        //Snackbar.make(getView(), (String) msg.obj,
Snackbar.LENGTH_SHORT).show();
        break;
    case Constants.MESSAGE_READ_FILE:
        Toast.makeText(getActivity(), "file read",
Toast.LENGTH_SHORT).show();
        Long currentTime = System.currentTimeMillis();
        messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(msg.obj + "",
                currentTime, Constants.MESSAGE_TYPE_FILE_RECEIVED,
                receivedUsername, true));
        chatListAdapter.notifyDataSetChanged();
        chatView.smoothScrollToPosition(messageList.size());
        break;
    case Constants.MESSAGE_WRITE_FILE:
        Toast.makeText(getActivity(), "file write",
Toast.LENGTH_SHORT).show();
        currentTime = System.currentTimeMillis();
        messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(
                msg.obj + "", currentTime,
                Constants.MESSAGE_TYPE_FILE_SENT,
                sentUsername, true));
        chatListAdapter.notifyDataSetChanged();
        chatView.smoothScrollToPosition(messageList.size());
    case Constants.MESSAGE_DEVICE_NAME:
        connectedDeviceBluetoothName =
msg.getData().getString(Constants.DEVICE_NAME);
        if (null != getActivity()) {
            Toast.makeText(getActivity(), "Connected to: " +
connectedDeviceBluetoothName, Toast.LENGTH_SHORT).show();
        }
        break;
    case Constants.MESSAGE_DEVICE_ADDRESS:
        connectedDeviceBluetoothAddress =
msg.getData().getString(Constants.DEVICE_ADDRESS);
        break;
    case Constants.MESSAGE_TOAST:
        if (null != getActivity()) {
            Toast.makeText(getActivity(),
msg.getData().getString(Constants.TOAST), Toast.LENGTH_SHORT).show();
        }
        break;
    }
}

};

private BluetoothAdapter mBluetoothAdapter = null;
private BluetoothChatService chatService = null;
private ChatRecyclerAdapter chatListAdapter;
private EditText inputField;
private ImageButton btnSend;
private ImageButton btnAttach;

```

```

        private Toolbar toolbar;
        private RecyclerView chatView;

        private final int FILE_SELECT_CODE = 1;
        private final String FILE_BROWSER_CACHE_DIR =
"YorickCache";

        protected final int PERMISSION_REQUEST_ENABLE_BLUETOOTH = 2;
        protected final int PERMISSION_REQUEST_CONNECT_DEVICE_SECURE = 3;
        protected final int PERMISSION_REQUEST_CONNECT_DEVICE_INSECURE = 4;

        private void createChannel(NotificationManager manager) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                NotificationChannel notificationChannel = new
NotificationChannel(CHANNEL_ID, CHANNEL_ID,
NotificationManager.IMPORTANCE_DEFAULT);
                manager.createNotificationChannel(notificationChannel);
            }
        }

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

            if (mBluetoothAdapter == null) {
                Toast.makeText(getActivity(), "Bluetooth is not available",
Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        public void onActivityCreated(@Nullable Bundle savedInstanceState) {
            super.onActivityCreated(savedInstanceState);

            mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

            if (mBluetoothAdapter == null) {
                Toast.makeText(getActivity(), "Bluetooth is not available",
Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                Bundle savedInstanceState) {
            View view = inflater.inflate(R.layout.fragment_chat, container, false);
            //chatService = new BluetoothChatService(getActivity(), handler
            dbHelper = new SQLiteDatabaseHelper(getActivity().getApplicationContext());

            LinearLayoutManager layoutManager = new
LinearLayoutManager(getActivity());

            messageList = new
ArrayList<com.kernelpanic.yorickmessenger.util.Message>();

            chatView = view.findViewById(R.id.messagesContainer);
            chatView.setLayoutManager(layoutManager);

```

```

        return view;
    }

    @Override
    public void onStart() {
        super.onStart();

        try {
            if (!mBluetoothAdapter.isEnabled()) {
                Intent enableBT = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                startActivityForResult(enableBT,
PERMISSION_REQUEST_ENABLE_BLUETOOTH);
                mBluetoothAdapter.enable();
                Toast.makeText(getActivity(), "Enabling BT adapter",
Toast.LENGTH_SHORT).show();
            } else if (chatService == null) {
                setupChat();
                Toast.makeText(getActivity(), "Initiating the scan devices
procedure", Toast.LENGTH_SHORT).show();
                Intent server = new Intent(getActivity(),
ScanListActivity.class);
                /*
                Just for 'difficult tech process' imitating.
                Joke. Made this to wait until the fragment change animation
will done
                */
                Handler postDelayedHandler = new Handler();
                postDelayedHandler.postDelayed(new Runnable()
{
                    @Override
                    public void run() {
                        startActivityForResult(server,
PERMISSION_REQUEST_CONNECT_DEVICE_SECURE);
                    }
                }, 1000);
            }
        } catch (NullPointerException ex) {
            Log.d(TAG, "We have caught an exception: " + ex.getMessage());
        }
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        inputField = view.findViewById(R.id.chatFieldInput);
        btnSend = view.findViewById(R.id.btnSend);
        btnAttach = view.findViewById(R.id.btnAttach);

        Long currentTime = System.currentTimeMillis();
        //messageList.add(new com.kernelpanic.yorickmessenger.util.Message("Test
Received",
            //, Constants.MESSAGE_TYPE_RECEIVED, username, profilePic));

        //messageList.add(new com.kernelpanic.yorickmessenger.util.Message("Test
Sent",
            //currentTime, Constants.MESSAGE_TYPE_SENT, username,
profilePic));
    }

    @Override

```

```

public void onResume() {
    super.onResume();

    // Performing this check in onResume() covers the case in which BT was
    // not enabled during onStart(), so we were paused to enable it...
    // onResume() will be called when ACTION_REQUEST_ENABLE activity
returns.
    if (chatService != null) {
        // Only if the state is STATE_NONE, do we know that we haven't
started already
        if (chatService.getState() == BluetoothChatService.STATE_NONE) {
            // Start the Bluetooth chat services
            chatService.start();
        }
    }
}

private void setupChat() {

    chatListAdapter = new ChatRecyclerAdapter(messageList,
getActivity());
    chatView.setAdapter(chatListAdapter);

    inputField.setOnEditorActionListener(writeListener);

    btnSend.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            View view = getView();
            if (null != view) {
                //EditText textField =
view.findViewById(R.id.chatFieldInput);
                String message = inputField.getText().toString();
                sendMessage(message);
            }
        }
    });

    btnAttach.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            View view = getView();
            if (null != view) {
                Intent filePickIntent = new
Intent(Intent.ACTION_OPEN_DOCUMENT);
                filePickIntent.setType("*/*");
                filePickIntent.addCategory(Intent.CATEGORY_OPENABLE);

filePickIntent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION |
Intent.FLAG_GRANT_WRITE_URI_PERMISSION);

                String FILE_PATH =
Environment.getExternalStorageDirectory().getAbsolutePath() + "/YMessenger/";
                Toast.makeText(getActivity(), FILE_PATH,
Toast.LENGTH_SHORT).show();
                File file = new File(FILE_PATH);

                if
(filePickIntent.resolveActivity(getActivity().getPackageManager()) != null) {

startActivityResult(Intent.createChooser(filePickIntent, "Select a File to
Upload"), FILE_SELECT_CODE);
                }
            }
        }
    });
}

```

```

        }
    }
});

chatService = new BluetoothChatService(getActivity(), handler);

//chatService = new BluetoothChatServiceClass(getActivity(), handler);
outStringBuffer = new StringBuffer(" ");
}

private TextView.OnEditorActionListener writeListener
    = new TextView.OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event)
{
    if (actionId == EditorInfo.IME_NULL && event.getAction() ==
KeyEvent.ACTION_UP) {
        String msg = v.getText().toString();
        sendMessage(msg);
    }
    return true;
}
};

private void sendMessage(String msg) {
    if (chatService.getState() != BluetoothChatService.STATE_CONNECTED) {
        Toast.makeText(getActivity(), "Not connected",
Toast.LENGTH_SHORT).show(); //TODO: add string to strings.xml
        return;
    }

    if (msg.length() > 0) {
        byte[] sendData = msg.getBytes();
        chatService.sendData(msg);

        outStringBuffer.setLength(0);
        inputField.setText(outStringBuffer);
    }
}

@Override
public void onActivityResult(int requestCode, int resultCode, @Nullable Intent
data) {
    Bitmap bitmap = null;
    switch (requestCode) {
        case PERMISSION_REQUEST_CONNECT_DEVICE_SECURE:
            if (resultCode == Activity.RESULT_OK && chatService.getState() !=
BluetoothChatService.STATE_CONNECTED) {
                connectToDevice(data);
                onConnectDataExchange();
            }
            break;
        case PERMISSION_REQUEST_CONNECT_DEVICE_INSECURE:
            if (resultCode == Activity.RESULT_OK && chatService.getState() !=
BluetoothChatService.STATE_CONNECTED) {
                connectToDevice(data);
                onConnectDataExchange();
            }
        case PERMISSION_REQUEST_ENABLE_BLUETOOTH:
            if (resultCode == Activity.RESULT_OK) {
                setupChat();
            } else {

```

```

        Log.d("YMessenger.Result()", "Bluetooth is not enabled");
    }
    case FILE_SELECT_CODE:
        if (resCode == Activity.RESULT_OK && data != null
            && chatService.getState() ==
BluetoothChatService.STATE_CONNECTED) {
            Uri uri = data.getData();
            Context context = getActivity();
            final String path;
            try {
                path = writeFileContent(uri);
                Toast.makeText(getActivity(), path,
Toast.LENGTH_SHORT).show();
                chatService.sendFile(path);
            } catch (IOException e) {
                e.printStackTrace();
            }
            /*
                final String path2 = RealPathHelper.getRealPath(context,
uri);
                Toast.makeText(context, "FileWizard path " + path1,
Toast.LENGTH_SHORT).show();
                Toast.makeText(context, "RealPathHelper path " + path2,
Toast.LENGTH_SHORT).show();
                Toast.makeText(context, "Uri " + uri,
Toast.LENGTH_SHORT).show();
                Long currentTime = System.currentTimeMillis();
                messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(path1, currentTime,
                Constants.MESSAGE_TYPE_FILE_RECEIVED, "username",
true));
                chatListAdapter.notifyDataSetChanged();*/
            /*
                try {
                    path = writeFileContent(uri);
                    Toast.makeText(getActivity(), path,
Toast.LENGTH_SHORT).show();
                    Long currentTime = System.currentTimeMillis();
                    messageList.add(new
com.kernelpanic.yorickmessenger.util.Message(path, currentTime,
                    Constants.MESSAGE_TYPE_FILE_RECEIVED,
"username", true));
                    chatListAdapter.notifyDataSetChanged();
                    chatService.sendFile(path);
                } catch (IOException e) {
                    e.printStackTrace();
                }*/
        }
    }
}

private String writeFileContent(Uri uri) throws IOException {
    InputStream inputStream =
getContext().getContentResolver().openInputStream(uri);
    if (inputStream != null) {
        final File yorickCacheDir = new File(getContext().getFilesDir(),
FILE_BROWSER_CACHE_DIR);
        Log.d(TAG, yorickCacheDir.getAbsolutePath());
        boolean isCacheDirExists = yorickCacheDir.exists();
        if (!isCacheDirExists) {
            isCacheDirExists = yorickCacheDir.mkdirs();
        }
        if (isCacheDirExists) {

```

```

        String filePath = yorickCacheDir.getAbsolutePath() + "/" +
getFileName(uri);
        OutputStream outputStream = new FileOutputStream(filePath);
        byte[] buffer = new byte[1024];
        int length;
        while((length = inputStream.read(buffer)) > 0) {
            outputStream.write(buffer, 0, length);
        }
        outputStream.flush();
        outputStream.close();
        return filePath;
    }
    inputStream.close();
}
return null;
}

private String getFileName(Uri uri) {
    String displayName = null;
    try (Cursor cursor = getContext().getContentResolver()
        .query(uri, null, null, null, null, null)) {
        if (cursor != null && cursor.moveToFirst()) {
            displayName =
cursor.getString(cursor.getColumnIndex(OpenableColumns.DISPLAY_NAME));
        }
    }
    return displayName;
}

private void connectToDevice(Intent data) {
    String deviceAddress = data.getExtras()
        .getString(ScanListActivity.EXTRA_DEVICE_ADDRESS);
    Toast.makeText(getActivity(), deviceAddress, Toast.LENGTH_SHORT).show();
    Toast.makeText(getActivity(), deviceAddress, Toast.LENGTH_LONG).show();
    BluetoothDevice device =
mBluetoothAdapter.getRemoteDevice(deviceAddress);
    chatService.connect(device);
}

private void onConnectDataExchange() {
    databaseUsername = dbHelper.getUserInfo();

    byte[] bytes = databaseUsername.getBytes();

    chatService.sendData(databaseUsername);
}
}

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

«Мобільний додаток для спілкування за допомогою технології Bluetooth.
Mobile Application for Communication via Bluetooth»

Виконав
ст. гр. КНТ-127

Я. А. Євдокімов

Керівник
Доцент, к.т.н,

О. О. Степаненко

Рисунок В.1 – Слайд 1

Мета, об'єкт, предмет дослідження
The purpose of the work, object, subject

- Мета роботи - створення мобільного додатку, що надає змогу користувачам приєднуватися до пристрою один одного та спілкуватися, використовую функції, що надає технологія Bluetooth.
- Об'єкт дослідження - програмні засоби для організації спілкування за допомогою технології Bluetooth.
- Предмет дослідження - мобільний додаток для реалізації організації спілкування за допомогою технології Bluetooth.
- Актуальність та проблематика роботи полягає в тому, що на сьогоднішній день не всі люди та не всюди можуть мати стабільний доступ до Інтернету, але виникає необхідність відправити якесь повідомлення, або файл.

2

Рисунок В.2 – Слайд 2

Цілі проектування

Goals

- функціонування застосунку;
- простий інтерфейс;
- наявність бази даних;
- опанувати технологію Bluetooth;
- удосконалити навички проектування android-застосунків та програмування на мові Java.

3

Рисунок В.3 – Слайд 3

Порівняння аналогів

Comparison of analogues

Назва застосунку	Привабливість та зручність інтерфейсу	Стабільність	Зрозумілість
Bluetooth Chat (Kotlin)	+	+	+
BluetoothChat	-	+	+
BlueChat (Kotlin)	+	-	+
Ymessenger	+	+	+

4

Рисунок В.4 – Слайд 4

Вибір засобів проектування

Development tools

- Java
- Android Studio
- Figma (Pictures, Icons, etc)

Мови програмування

#	Назва
1	Dart (Flutter Framework)
2	C# (Xamarin Framework)
3	Kotlin

Середі розробки

#	Назва
1	MS Visual Studio
2	Unreal Eng.
3	Unity3D
4	Eclipse IDE
5	Xcode

Проектування дизайну застосунку

#	Назва
1	Adobe Fireworks
2	Adobe XD
3	inVision

5

Рисунок В.5 – Слайд 5

Складнощі під час проектування. Різноманітність версій Android

Difficulties during the development

Назва	Номер	Рівень API	Рівень NDK
Cupcake	1.5	3	1
Donut	1.6	4	
Eclair	2.0-2.1	5	2
	2.0.1	6	
	2.1	7	3
Froyo	2.2.x	8	4
Gingerbread	2.3-2.3.2	9	
	2.3.3-2.3.7	10	5
	3.0	11	
Honeycomb	3.1	12	6
	3.2.x	13	
Ice Cream Sandwich	4.0.1-4.0.2	14	7

Продовження таблиці			
Ice Cream Sandwich	4.0.3-4.0.4	15	8
Jelly Bean	4.1-4.3	16-18	
KitKat	4.4-4.4.4	19	
Lollipop	5.0-5.1	21-22	
Marshmallow	6.0	23	
Nougat	7.0-7.1	24-25	
Oreo	8.0-8.1	26-27	
Pie	9	28	
Android 10 (Q)	10	29	
Android 11 (R)	11	30	

* - Android версії Q та R викликають труднощі при розробці

6

Рисунок В.6 – Слайд 6

Основні моменти. Підключення до пристрою

Main part. Connection to the another remote Bluetooth device

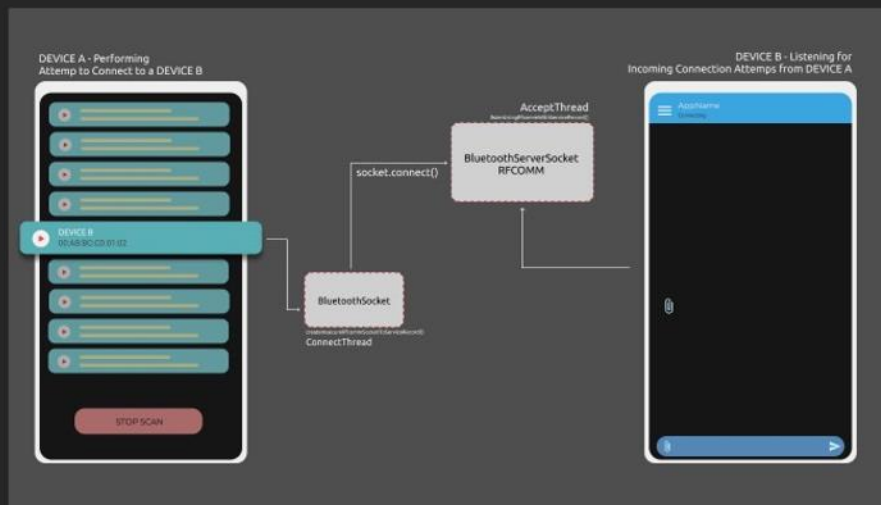


Рис 4 – Device A connecting to Device B

9

Рисунок В.9 – Слайд 9

Основні моменти. Відправка даних

Main part. Data exchange

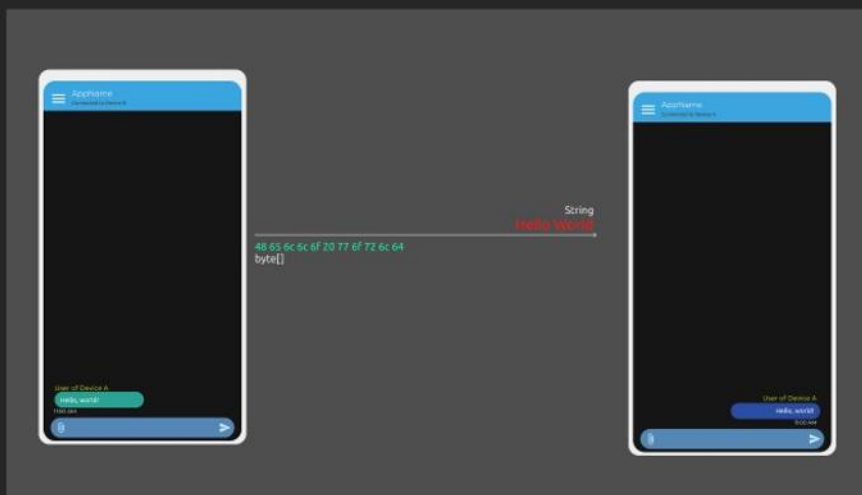


Рис 5 – Device A sending data to Device B

10

Рисунок В.10 – Слайд 10

Тестування застосунку

Testing part

№	Виробник	Модель	Версія Android	Оболонка	Результати перевірок
1	Xiaomi	Mi A1	10	crDroid (Custom Unofficial)	Проблеми з адаптивністю інтерфейсу
2	Xiaomi	Mi Note 10 Lite	11	MIUI	Критичні проблеми при роботі з файловою системою
3	OnePlus	6T	10	Oxygen OS	Незначні та нестабільні помилки при роботі з Bluetooth. Допомогає перезапуск пристрою
4	Huawei	MediaPad T3 10	7	EMUI	Проблеми з адаптивністю – планшет

13

Рисунок В.13 – Слайд 13

Висновки

Summary

Основні реалізовані функції додатку:

- Сканування області Bluetooth для пошуку віддалених пристроїв.
- Підключення одного пристрою до іншого.
- Створення користувацького профілю.
- Обмін користувацькими даними.
- Відправлення та приймання повідомлень файлів.
- Відправлення та приймання текстових повідомлень
- Світла та темна теми.

Шляхи удосконалення:

- реалізація обміну файлами на всіх версіях Android;
- поліпшення адаптивності інтерфейсу на пристроях з маленькими діплеями;
- реалізація підтримки групових підключень на три чи більше осіб.

14

Рисунок В.14 – Слайд 14

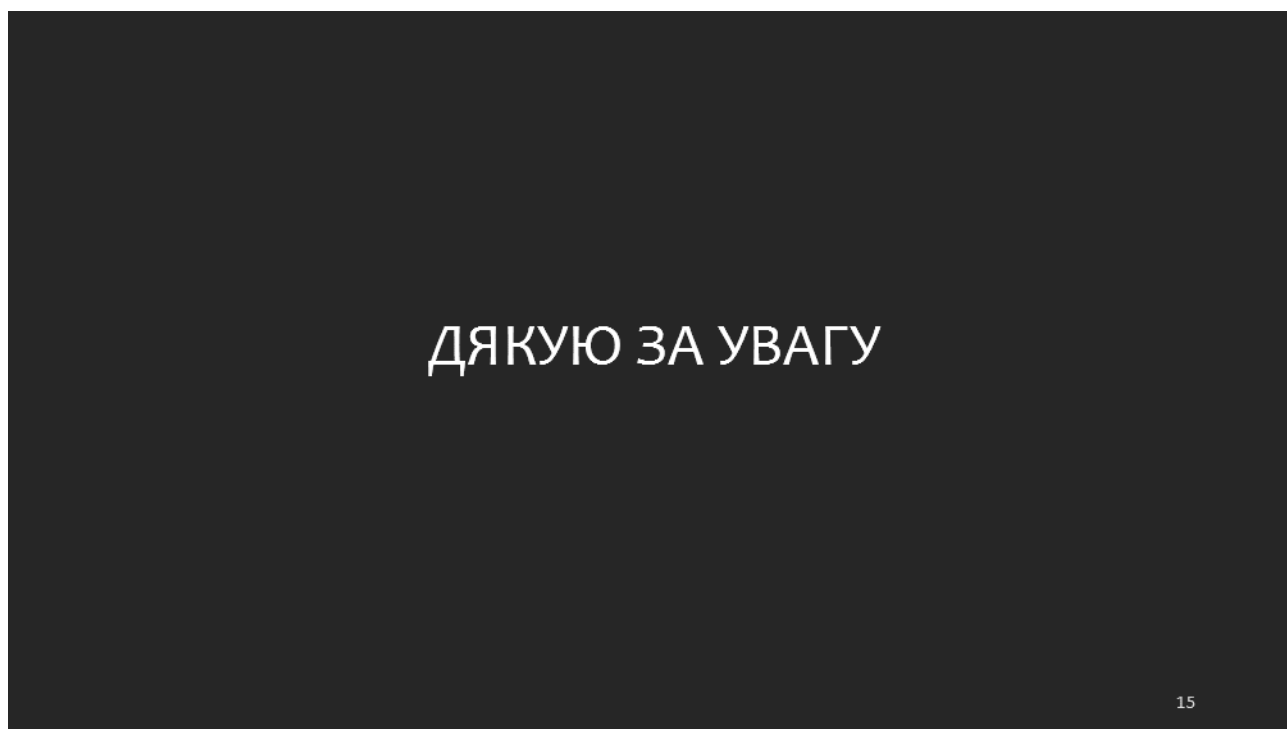


Рисунок В.15 – Слайд 15