

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет**

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Наталія АУШЕВА

« ___ » _____ 2022 р.

Дипломна робота

на здобуття ступеня бакалавра

спеціальності 122 «Комп'ютерні науки»

освітня програма «Комп'ютерний моніторинг та геометричне
моделювання процесів і систем»

на тему: «Месенджер в режимі реального часу для он-лайн курсу "Структури даних"»

Виконав:

студент IV курсу, групи ТР-81

Мусьол Євгеній Геннадійович _____

Керівник:

доцент, кандидат технічних наук

Кузьменко Ігор Миколайович _____

Рецензент:

доцент, кандидат технічних наук _____

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студент _____

Київ – 2022

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший, бакалаврський

Напрямок підготовки: 122 “Комп’ютерні науки”

Спеціалізація: Комп’ютерний моніторинг та геометричне моделювання процесів і систем

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Наталія АУШЕВА
(підпис)

” ____ ” _____ 2022р.

ЗАВДАННЯ

на дипломну роботу студенту

Мусьол Євгеній Геннадійович

(прізвище, ім’я, по батькові)

1. Тема роботи: Месенджер в режимі реального часу для он-лайн курсу "Структури даних"

керівник роботи: Кузьменко Ігор Миколайович, доцент, кандидат технічних наук
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від “25” травня 2022 р. № _____

2. Строк подання студентом роботи: 10 червня 2022 р.

3. Вихідні дані до роботи: мова програмування Java/Kotlin, середовище розробки IntelliJ IDEA 2020, СКБД PostgreSQL

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): реалізувати систему повідомлень, розробити функціонал соціальної взаємодії, провести дослідження та порівняння серед існуючих рішень у схожих продуктах з темою дипломної роботи.

5. Перелік ілюстративного матеріалу: зображення додатку, зображення аналогів, «Висновки».

6. Дата видачі завдання «20» лютого 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.05.2022 р.	
2.	Вивчення та аналіз задачі	03.05.2022-09.05.2022 р.	
3.	Аналіз аналогічних продуктів	10.05.2022-16.05.2022 р.	
4.	Вибір систем та методів розробки	17.05.2022-23.05.2022 р.	
5.	Проектування, розробка та тестування програмного продукту	10.05.2022-28.05.2022 р.	
6.	Оформлення пояснювальної записки	28.05.2022-02.06.2022 р.	
7.	Захист програмного продукту		
8.	Передзахист	08.06.2022 р.	
9.	Захист		

Студент

(підпис)

Муцьол Є. Г.

(прізвище та ініціали)

Керівник роботи

(підпис)

Кузьменко І. М.

(прізвище та ініціали)

АНОТАЦІЯ

Метою дипломної роботи є розробка месенджера для обміну повідомленнями в реальному часі для операційної системи Android. Аналіз необхідних особливостей, недоліків та переваг аналогічних рішень на ринку. Результат роботи програми може бути використаний для полегшення обміну інформацією для он-лайн курсів, а саме “Структури даних”.

Дипломна робота має обсяг 47 аркушів, містить 1 додаток і 20 посилань. Також наведено 22 рисунків.

Ключові слова: розробка месенджера, СКБД, клієнт-сервер, он-лайн, дані.

В результаті роботи був розроблений месенджер, за допомогою якого можна надсилати/отримувати повідомлення в приватних або групових чатах.

ANNOTATION

The purpose of the thesis is to develop a messenger for real-time messaging on the Android operating system. Analysis of required features, advantages and disadvantages of popular similar existing solutions. The result program can be used to make easier information exchange for online courses, especially for “Data Structures”.

Thesis has a volume of 47 sheets and contains 1 appendices and 20 references. There are also 22 figures.

Key words: messenger development, DBMS, client-server, online, data.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	7
ВСТУП.....	8
1. ЗАДАЧА МЕСЕНДЖЕРА ПО ОБМІНУ ПОВІДОМЛЕНЬ.....	11
2. АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ.....	13
2.1 Telegram.....	13
2.2 Viber.....	15
2.3 WhatsApp.....	17
3. ЗАСОБИ РОЗРОБКИ.....	20
3.1 IntelliJ IDEA.....	20
3.2 Java.....	21
3.3 Kotlin.....	21
3.4 Android Studio.....	22
3.5 Postman.....	23
4. ПРОЕКТУВАННЯ ДОДАТКУ.....	24
4.1. Архітектура побудови додатку.....	24
4.2 Концептуальна модель. Діаграма потоків даних.....	25
4.3 Інфологічна модель. Діаграма сутностей.....	29
4.4 Нормалізація моделі.....	31
4.5 Структура програми.....	31
4.6 Реалізація взаємодії з базою даних.....	32
5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ.....	34
ВИСНОВКИ.....	44

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	46
ДОДАТОК А.....	48

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД — Бази даних

СКБД — Система керування базами даних

SQL — Structured Query Language

Мобільний додаток — програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях.

IDE — комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм.

SDK — набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати прикладні програми за визначеною технологією або для певної платформи (програмної або програмно-апаратної).

ВСТУП

В наш час дуже швидко розвиваються різноманітні технології, людство накопичує знання і максимально їх використовує для поліпшення якості життя та зменшення використання різноманітних ресурсів у всіх сферах життя.

Велику частину часу, люди проводять використовуючи свої гаджети. А саме - пошук інформації, різноманітний розважальний контент та спілкування он-лайн. І саме спілкування здійснюється за допомогою соціальних мереж та месенджерів. Месенджер - це програмний продукт ціль якого, спростити взаємодію двох або більше користувачів за допомогою текстових, звукових, або відео-повідомлень. А різниця між існуючими готовими месенджерами заключається в: графічному інтерфейсі, типу зв'язку, оптимізації доставки та шифруванню повідомлень. Також у багатьох галузях намагаються максимально автоматизувати та зробити набагато простішими для керування людьми усі процеси шляхом використання зручних та багатофункціональних програмних застосунків. Будь яка система за свою мету має роботу з даними, їх обробку та візуалізацію. Тому для будь-якого програмного застосунку важливою є робота з базою даних: правильне проектування, оптимізація запитів та коректна обробка отриманих даних. Метою проекту є закріплення навичок з роботи над базами даних та проектування програмного застосунку з використанням бази даних для реалізації системи реального часу обміну повідомленнями. Об'єктом дослідження є проектування баз даних, виконання запитів до бази даних та обробка цих запитів. Методами дослідження є об'єктно орієнтоване програмування та теорія баз даних. Завданням дипломної роботи є розробка месенджеру з використанням реляційної бази даних для збереження інформації та реалізацією методів взаємодії бази даних з графічним інтерфейсом. Інструментами розробки є мови програмування Java, Kotlin, Spring Boot Framework, Android SDK та середовища програмування IntelliJ IDEA, Android Studio. Для оформлення було використано он-лайн текстовий редактор Google Docs, для малювання діаграм - додаток draw.io.

У результаті створено сервіс з можливістю аутентифікації та обміну повідомлень в приватних або групових чатах.

1 ЗАДАЧА МЕСЕНДЖЕРА ПО ОБМІНУ ПОВІДОМЛЕНЬ

У рамках дипломної роботи необхідно реалізувати систему повідомлень, розробити функціонал соціальної взаємодії, провести дослідження та порівняння серед існуючих рішень у схожих продуктах з темою дипломної роботи.

Для виконання цілі потрібно вирішити наступні задачі:

- Провести аналіз готових рішень;
- Провести аналіз, огляд та пояснення до інструментальних засобів, використаних у розробці програмного продукту;
- Спроекувати додаток за допомогою DFD 0 моделі та “сутність – зв’язок” діаграми;
- Організувати взаємодію між клієнт-додатком та сервером.

Головним призначенням даного програмного продукту є розробка додатку по обміну текстовими повідомленнями, створення облікових записів та подальшим використанням на платформах ARM (Android) .

Дана система може бути застосована в навчальних цілях, для поліпшення та покращення обміну інформацією між студентами та викладачами, адже додаток може бути використаний на мобільних пристроях.

При завантаженні додатку пропонує створити новий або увійти в обліковий запис. Так присутня БД з системою аутентифікації та збереженням даних користувачів та чатів, для подальшого завантаження на клієнт додатку. Також можливе додавання інших облікових записів у “список друзів”.

Реалізація даної системи включає такі підзадачі: розробка концептуальної моделі, виокремлення сутностей, реалізація атрибутів сутностей та налаштування зв’язків між ними, побудова інфологічної моделі, Проектування класів та сервісів відповідно до моделі, створення методів взаємодії з сутностями відповідно до концептуальної моделі системи, розробка мобільного додатку для взаємодії з

програмою та контролерів для налагодження комунікації між мобільним додатком та сервісами програми, тестування програми.

2 АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ

Безперечно, одною, якщо і не головною, перевагою використання месенджеру є зберігання повідомлень, та пошук необхідної для користувача інформації в будь-який момент часу. В зв'язку з цим, актуальність телефонних дзвінків зменшується, і є вірогідність, що в недалекому майбутньому їх замінять месенджери.

Актуальність того, чи іншого месенджеру полягає у шифруванні повідомлень, за допомогою закритих ключів. В зв'язку з цим, користувачі, які не мають цих ключів, не зможуть отримати інформацію, яка передається на сервер або клієнт. В кожному додатку є свої плюси та мінуси.

2.1. Telegram

Telegram — кросплатформений додаток для миттєвої передачі та отримання повідомлень. Перший запуск відбувся в серпні 2013-го року. Авторами додатку є Микола та Павло Дурові. Сама клієнтська частина написана на високорівневій мові програмування C++. Для неї створили протокол MTProto, який використовує декілька протоколів шифрування. При аутентифікації користувача також використовуються алгоритми шифрування RSA-2048 та DH-2048. Також для перевірки використовують криптографічні хеш-алгоритми SHA-1 та MD5.

В самому месенджері, крім відправки звичайних повідомлень, є функціонал здійснення телефонних викликів, в тому числі і шифрованих, а також боти та групові чати.

Унікальність додатку полягає в тому, що при авторизації на новому пристрої не потрібно завантажувати та зберігати резервні файли повідомлень, все відбувається автоматично. Вся історія повідомлень завантажується при відкритті чату, а файли зберігаються на іншому сервері та доступні в будь-який момент.

Також підтримуються бесіди, в яких одночасно можуть бути до 50000 користувачів.

Для того, щоб додати іншого користувача в групу, або просто відправити йому повідомлення, не обов'язково потрібно знати його номер телефону. Адже в продукту реалізована можливість встановлення унікального нікнейму, який можна задати в налаштуваннях профіля.

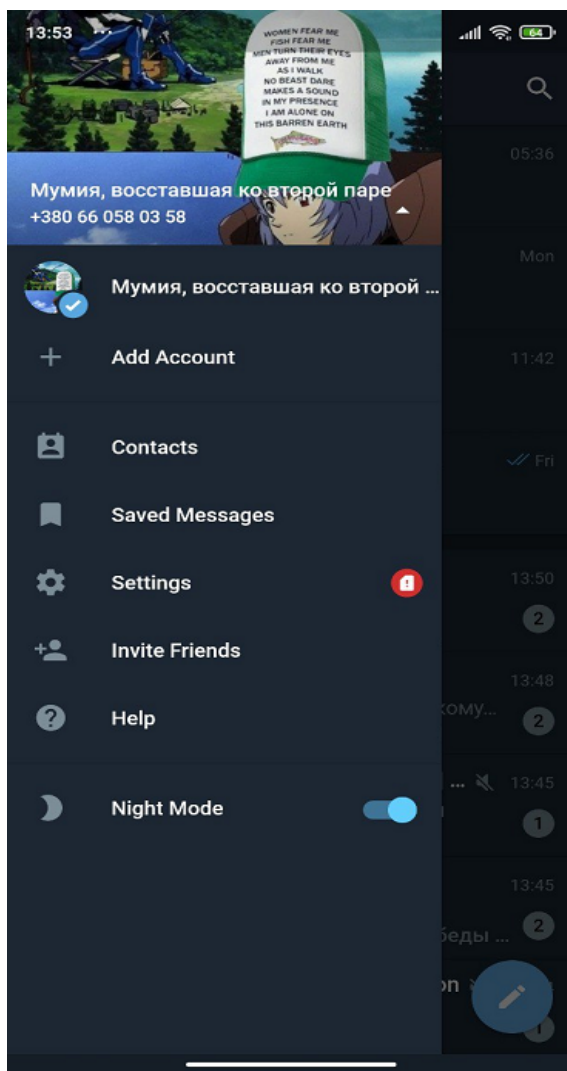


Рисунок 2.1 —Інтерфейс користувача Telegram на платформі Android

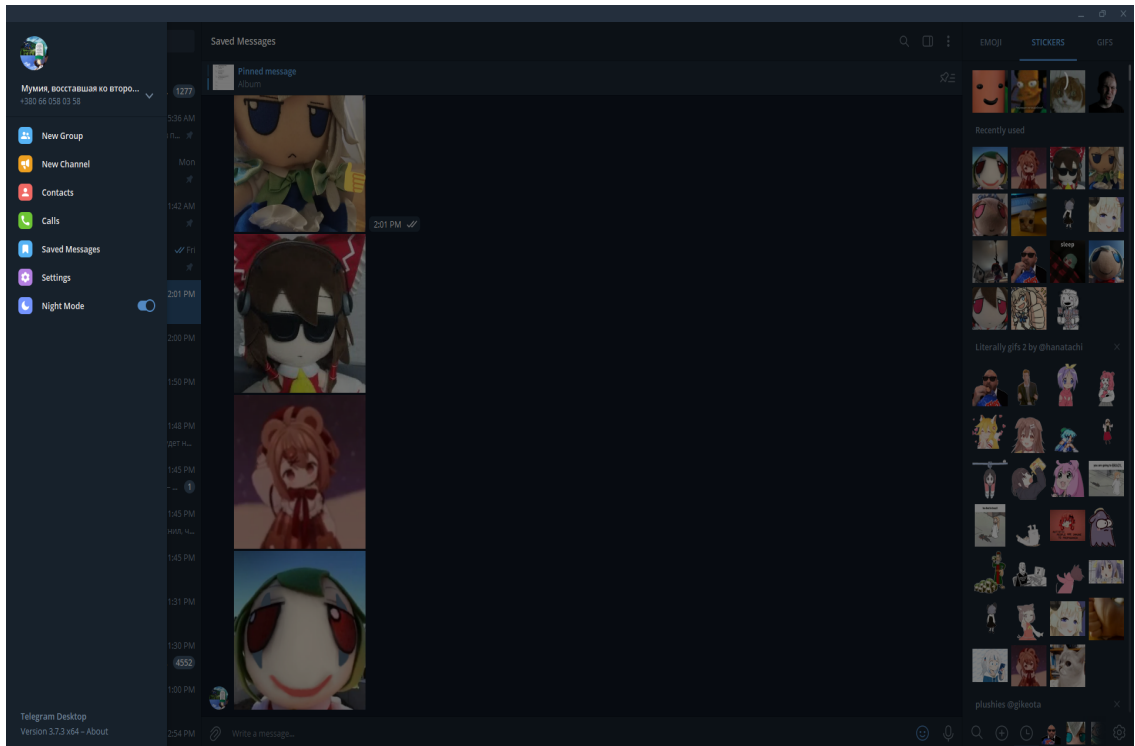


Рисунок 2.2 —Інтерфейс користувача Telegram на платформі Windows

Ще одним плюсом даного додатку є безкоштовні стікери, які може створити будь-який користувач.

2.2 Viber

Viber — кросплатформений додаток, який дозволяє здійснювати дзвінки та проводити обмін повідомленнями через WiFi та мобільну мережу. Також існує додаткова платна функція ViberOut, яка дозволяє здійснити дзвінок на пристрій без інтернету, або на якому навіть не встановлено додаток. Авторами цієї програми є Ігор Магазіннік та Мальмон Марко. Перший запуск додатку відбувся в грудні 2010-го року. Для розробки було використані такі мови програмування як: Java, C, Python, C++, Objective-C.

У самому додатку є шифрування повідомлень, секретні чати, дзвінки. Також, на даний момент, розробники ввели відеодзвінки, але лише на бета-версія продукту. Також є безліч стікерів як платних, так і безкоштовних.

Viber являє собою корпоративним месенджером, адже великі мережі магазинів, у яких є ваші дані в базі, можуть надсилати вам рекламу про будь-які свої

акції. Комусь це може здатися практичним, іншим - дуже набридливим. В будь-якому разі - цей функціонал можна легко вимкнути в налаштуваннях.

В додатку також доступні багатокористувацькі чати, боти та публічні групи.

Хоча є один суттєвий мінус - реклама, яка створена на основі ваших уподобань. Додаток зберігає та надсилає всі ваші дані активності на сервер та підбирає рекламні афіші спеціально для Вас.

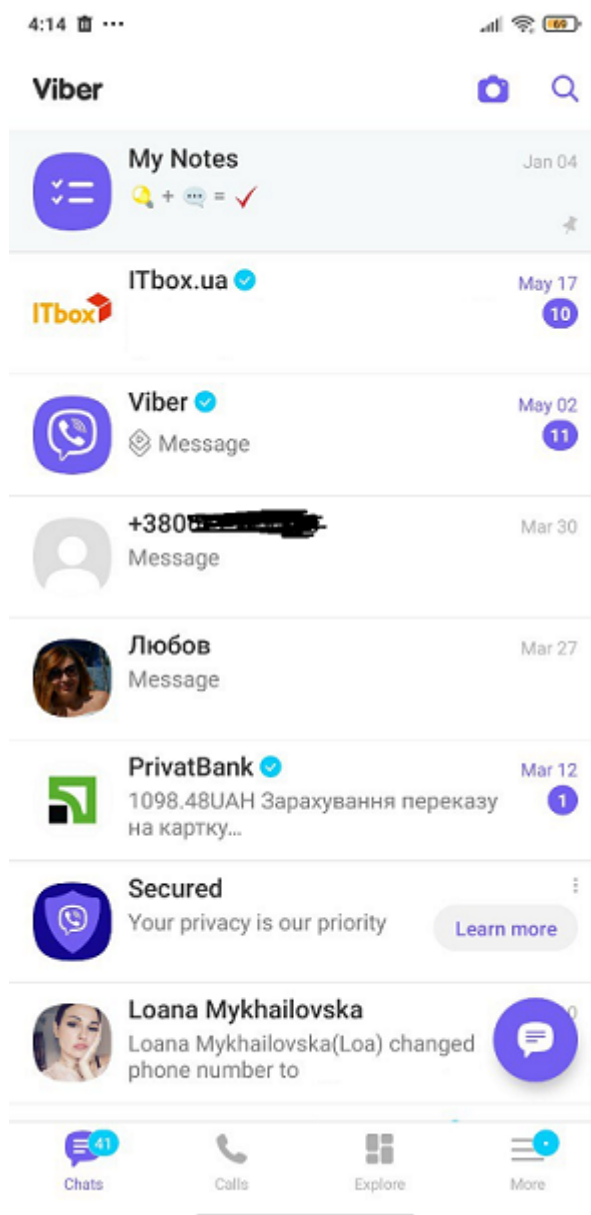


Рисунок 2.3 — Інтерфейс користувача Viber на платформі Android

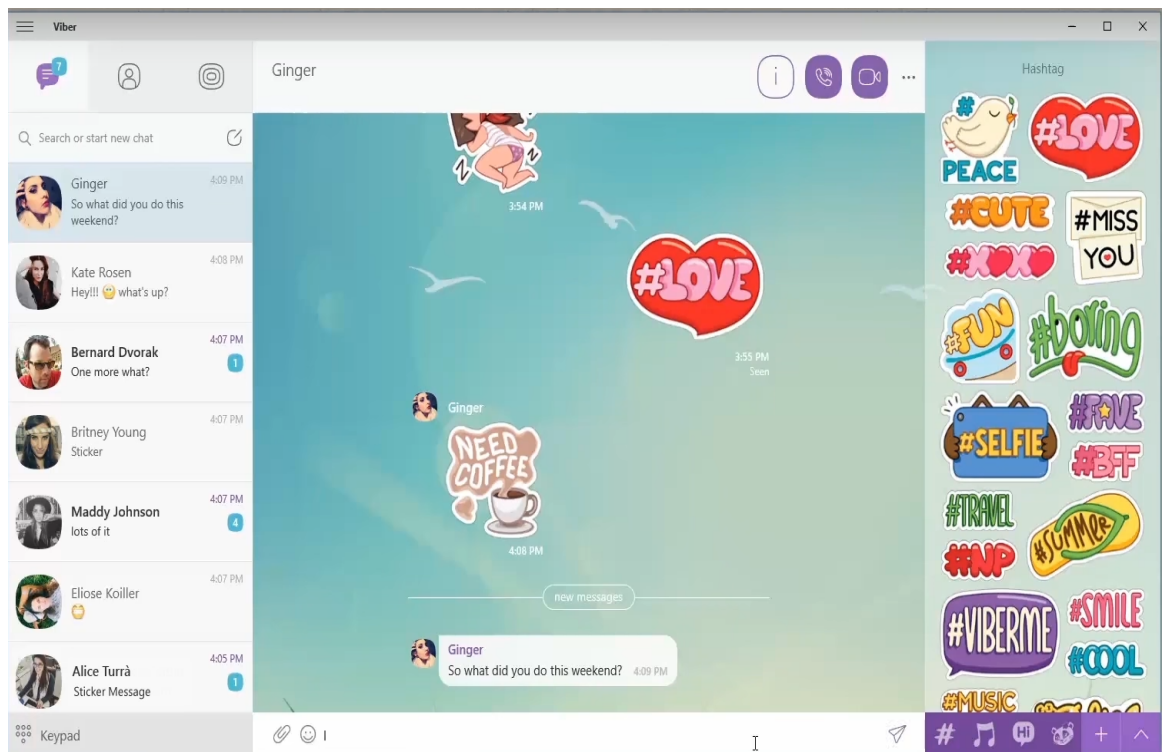


Рисунок 2.4 — Інтерфейс користувача Viber на платформі Windows

Зв'язатися з необхідним користувачем можна за номером мобільного телефону, які додаток автоматично синхронізує з телефонної книги.

2.3 WhatsApp

WhatsApp — популярна безкоштовна система миттєвого обміну текстових повідомлень для мобільних та інших систем з підтримкою голосового та відеозв'язку. Компанія WhatsApp була куплена компанією Facebook в жовтні 2014-го року. Сам клієнтський додаток був створений за допомогою мови програмування Erlang. Перша версія програми була створена в 2009-му році.

WhatsApp є наймасовішим месенджером у світі. За оригінальною задумкою додаток повинен був бути на платній основі: перший рік використання надавався всім користувачам безкоштовно, в той час як дальніше користування месенджером коштувало приблизно 1 USD за рік.

WhatsApp використовує модифікований протокол шифрування повідомлень Extensible Messaging and Presence Protocol (XMPP). При завантаженні та першому запуску додатку, створюється аккаунт на сервері s.whatsapp.net, у якому

використовується мобільний номер телефону користувача як його логін. Мобільна версія месенджеру автоматично використовує в якості пароля від облікового запису, MD5-хеш суму від видозміненого ідентифікатору IMEI, в той час як версія на базі IOS використовує MD5-хеш суму від MAC-адресу.

Додаток автоматично завантажує та додає список контактів з телефонної книги пристрою. Це можливо через той факт, що всі користувачі мережі реєструють облікові записи за допомогою номеру мобільного телефону.

Також в WhatsApp є можливість зміни “статусу”, в якому може міститися текст, фото, або невелике за розміром відео, яке буде автоматично буде видалено через 24 години від його встановлення. Вперше, такий функціонал з'явився в іншому продукті в Facebook - Instagram.

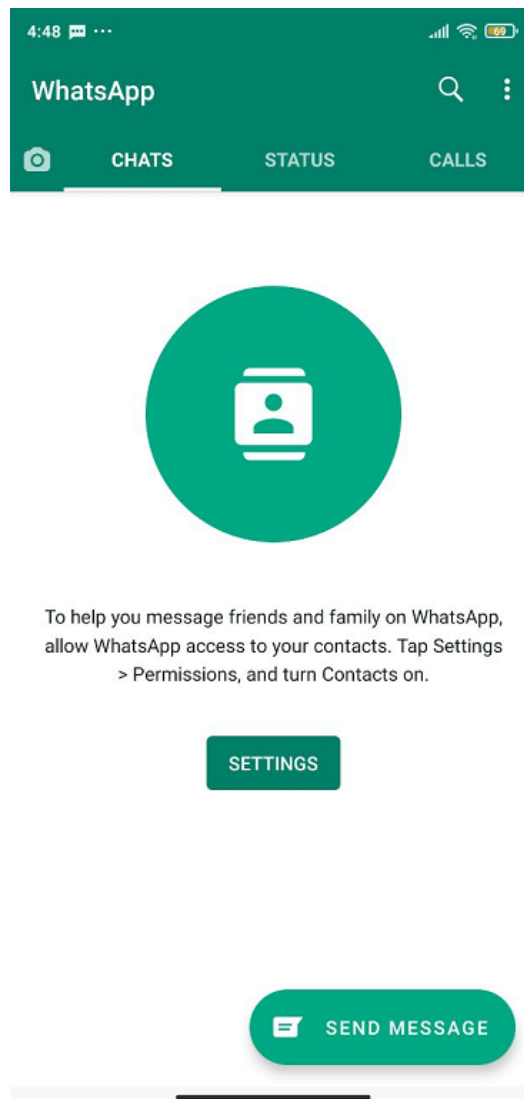


Рисунок 2.5 — Інтерфейс користувача WhatsApp на платформі Android

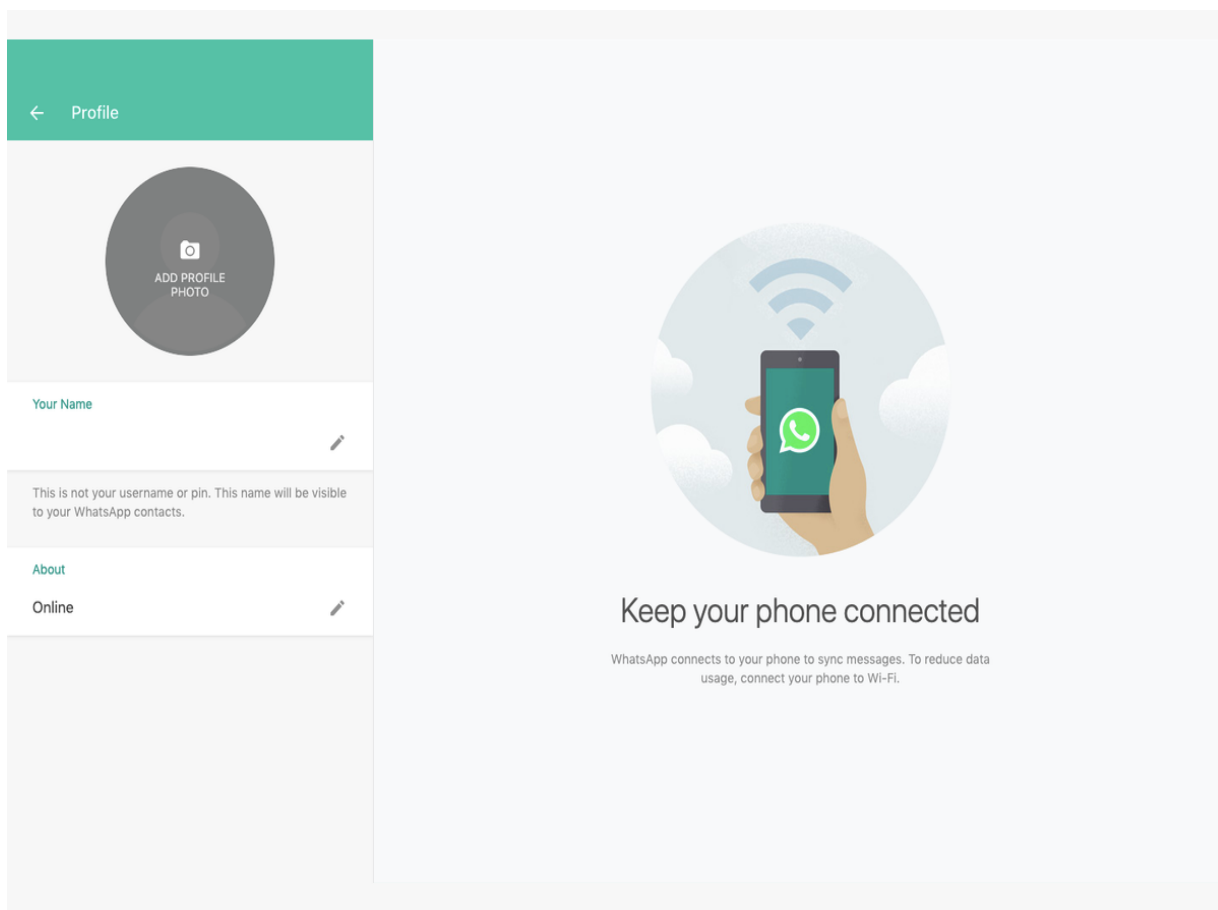


Рисунок 2.6 — Інтерфейс користувача WhatsApp на платформі Windows

У версії для персонального комп'ютеру та на веб версії є особливість, яка заключається в тому, що система не може зв'язатися з серверами месенджера, якщо на мобільному пристрої немає підключення до WiFi або мобільної мережі. Також не можна бути одночасно активним в веб/ПК та мобільній версіях.

3 ЗАСОБИ РОЗРОБКИ

Для розробки месенджера було використано декілька мов програмування, а саме: Java, Kotlin. В якості інструмента для розробки було обрано рішення від JetBrains, а саме IntelliJ IDEA 2020. Для операцій по створенню та зберіганню інформації в базі даних було використано СКБД PostgreSQL.

В наш час технологічні рішення прийнято вибирати виходячи з факту найкращої сумісності. Тому Java і Kotlin були вибрані в зв'язку з простотою в використанні та схожості. А IntelliJ IDEA є основною цільовою IDE для вибраних мною мов програмування. Для тестування серверної частини та відправки запитів було використана програма під назвою Postman, а для клієнтської частини - вбудований емулятор мобільного пристрою Android Studio. Також, для тестування готового продукту були використані доступні при розробці мобільні пристрої.

3.1. IntelliJ IDEA

IntelliJ IDEA - комерційне інтегроване середовище розробки для різних мов програмування (Java, Kotlin, Scala, PHP та інш.) від компанії JetBrains. Воно складається з редактору коду та компілятора. В редакторі пишеться код програми, а компілятор транслює його в байт код. Програма має зручний графічний інтерфейс, інструменти знаходяться у верхній частині відкритого вікна. Програма має всі необхідні функції для розробки невеликих проектів та безкоштовна для некомерційного використання при «Community Edition», та комерційного використання при «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію.

Тобто, середовище IntelliJ IDEA 2020 ідеально підходить для реалізації серверної частини програмного забезпечення даної дипломної роботи.

3.2. Java

Мова Java — об'єктно-орієнтована мова програмування, випущена 1995 року "Sun Microsystems" як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

Розробляючи мову спеціалісти керувались п'ятьма завданнями, які були перетворені в наступні принципи:

- Простота в використанні, об'єктна орієнтованість і легкість вивчення;
- Надійність і безпечність;
- Незалежність від архітектури;
- Можливість інтерпретації;
- Інтерактивність і динамічність.

3.3. Kotlin

Мова Kotlin - статично типізована мова програмування, що є похідною від Java і розробляється компанією JetBrains. Так само, як і Java компілюються у байт-код, який при виконанні віртуальною Java машиною для конкретної платформи

Перевагами Kotlin є такі аспекти:

- Краща типобезпечність.
- Лаконічний синтаксис.
- Можливість компіляції в нативний код платформи.
- Краща підтримка мови з боку компанії Google, що виражається у простоті написання мобільних додатків.
- Краща підтримка мови середовищем розробки мобільних додатків Android Studio.

Отже, мова програмування Kotlin підходить для реалізації клієнтської частини додатку.

3.4. Android Studio

Android Studio - це кроссплатформене інтегроване середовище розробки (IDE) для операційної системи Android. Сама IDE була розроблена компанією Google та написане на мові Java. IDE основана на програмному забезпеченні IntelliJ IDEA, та є офіційним засобом розробки від компанії JetBrains.

Особливостями Android Studio є:

- Перегляд макета на декількох конфігураціях екрану;
- Вбудована система сигнатур додатків;
- Підтримка Android Wear та Android TV;
- Шаблони основних макетів та компонентів;
- Статичний аналізатор коду, який знаходить проблеми з оптимізацією продуктивності;
- Перепроєктування коду без зміни функціоналу додатку;
- Генерація декількох арк-файлів.

Одним з ключових елементів Android Studio є наявність вбудованого модулю тестування - Android SDK. Він дозволяє встановити необхідну версію операційної системи та розширення екрану. При запуску проекту, він дозволяє вибрати пристрій, на якому планується надалі перегляд додатку, потім компілює проект та встановлює згенерований арк файл на пристрій. Також є функціонал підключення до реального пристрою по USB зв'язку та протестувати продукт вже на ньому.

3.5. Postman

Postman є найповнішою інструментальним додатком для розробки API. За допомогою цього інструменту можна створювати та редагувати HTTP-запити будь-якої складності. Створені запити автоматично зберігаються для їх повторного запуску та використання. Всі дані, такі як результати відповідей від серверу, також можна зберегти як файли. Таким чином, цей додаток економить купу часу для веб-розробників та ручним тестувальникам.

4 ПРОЕКТУВАННЯ ДОДАТКУ

Модель - це деякий аналог системи, аналіз та дослідження якого являється засобом отримання інформації про задану систему; абстрактне уявлення деякого реального процесу, пристрою або глобальної концепції.

Моделювання - дослідження системи на присутній їй моделях; побудова та дослідження аналогів реально існуючих об'єктів, процесів або явищ з метою отримання опису характеру поведінки цих явищ, а також для передбачення подальших наслідків та подій, які будуть отримані при використанні певної моделі.

4.1. Архітектура побудови додатку

В наш час, майже всі додатки мають архітектуру “Клієнт-Сервер”. Дана архітектура має два рівні - сам клієнт та сервер. При цьому є чітко визначені задачі для постачальника та замовника послуг. Клієнт та сервер можна рахувати як окремі програмні додатки. Зазвичай, ці два компоненти знаходяться на різних системах (комп'ютерах) та взаємодіють між собою за допомогою мережі та мережевих протоколів. Додатки-”постачальники” очікують від додатків-”замовників” запити та надають їм свої ресурси у вигляді даних або сервісних функцій. Так як, одна серверна програма може виконувати та обробляти запити від багатьох клієнтських додатків, її, зазвичай, встановлюють на спеціально виділеному комп'ютері з високою продуктивністю.

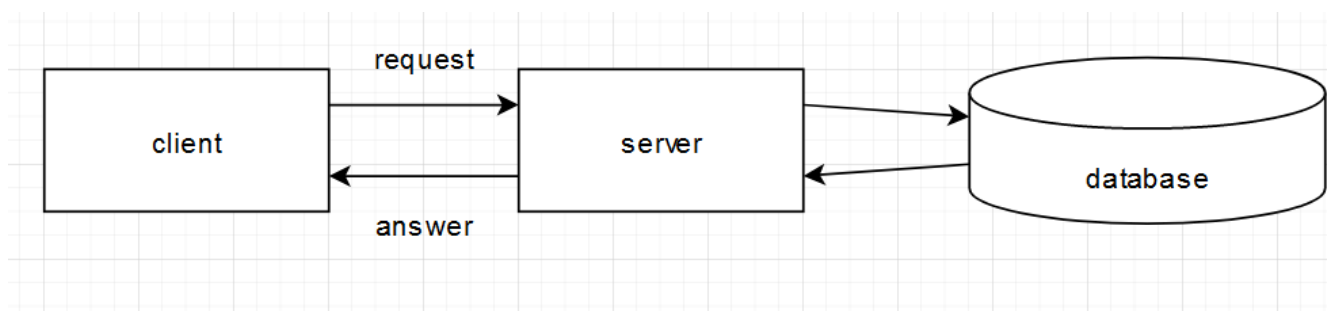


Рисунок 4.1 — Двошарова архітектура “Клієнт-Сервер”

У такої схеми є свої плюси та мінуси.

Плюси:

- Немає повторного коду сервера на клієнтах;
- Немає необхідності у великій потужності на клієнтських комп'ютерах, так як основна частина виконується на сервері;
- Зберігання та обробка даних виконується на сервері.

Мінуси:

- Незаплановане відключення серверу призводить до непрацездатності всієї клієнтської частини;
- Висока вартість обладнання.

В такому випадку існує модифікована версія архітектури, а саме тришарова схема “Клієнт-Сервер-Сервер”.

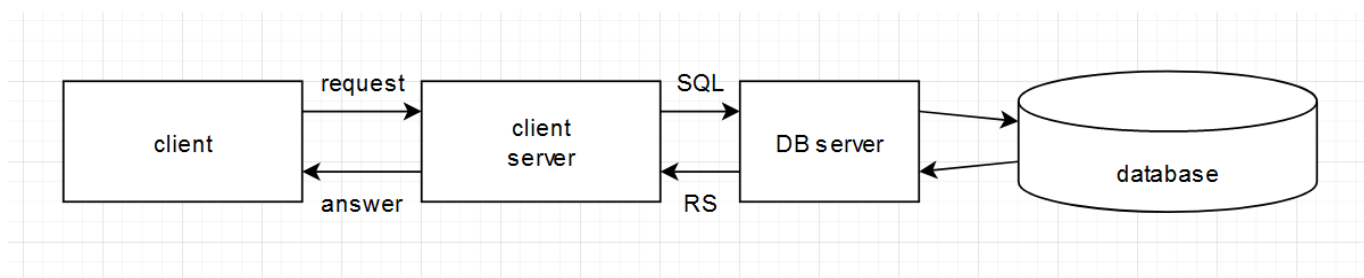


Рисунок 4.2 — Схема системи автоматичного регулювання

В такому випадку, сервер поділяється на дві частини - шар доступу до даних, та шар логіки. В першому випадку код містить дії з базою даних: збір, створення, оновлення, видалення, тощо. В другому випадку - функції, які необхідні для взаємодії з клієнтом.

4.2 Концептуальна модель. Діаграма потоків даних.

Для опису процесів на концептуальній моделі було використано діаграму потоків даних (Data Flow Diagram).

DFD - це діаграма потоків даних, спосіб представлення процесів обробки

інформації. Сам спосіб був розроблений незалежно від IDEF0. Ця методика, на відміну від IDEF0 не стандартизована.

Подібно до IDEF0, DFD представляє систему як мережу процесів, пов'язаних між собою за допомогою стрілок.

Безпосередньо DFD нотація складається з наступних елементів:

- Процес, тобто функція або послідовність дій, які потрібно зробити, щоб дані були оброблені. Це може бути створення замовлення, реєстрація клієнта в базі, тощо. У назвах процесів прийнято використовувати дієслова.
- Зовнішні сутності. Це будь-які об'єкти, які не входять у саму систему, але є для неї джерелом інформації чи одержувачами будь-якої інформації із системи після обробки даних. Це може бути людина, зовнішня система, будь-які носії інформації та сховища даних.
- Сховище даних. Внутрішнє сховище даних для процесів у системі. Дані, які були отримані перед обробкою, результати цієї інформації після обробки, а також проміжні значення повинні десь зберігатися. Для цього і використовують бази даних, таблиці чи будь-який інший варіант організації та зберігання даних. Тут зберігаються дані про клієнтів, заявки клієнтів та будь-які інші дані, що надійшли до системи або є результатом обробки процесів.
- Потік даних. У нотації відображається у вигляді стрілок, які показують, яка інформація входить, а яка виходить із того чи іншого блоку на діаграмі.

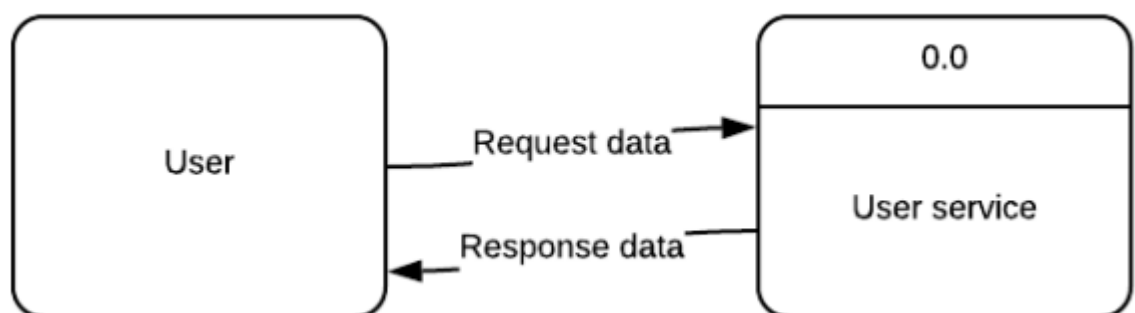


Рисунок 4.3 — Діаграма потоків даних. Система даних користувачів.

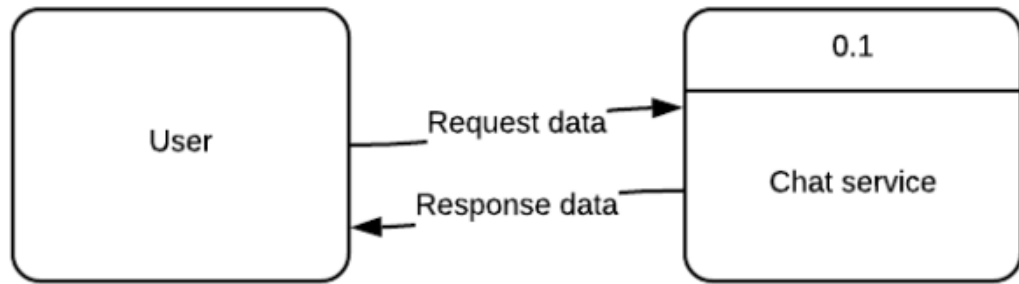


Рисунок 4.4 — Діаграма потоків даних. Система обробки чатів.

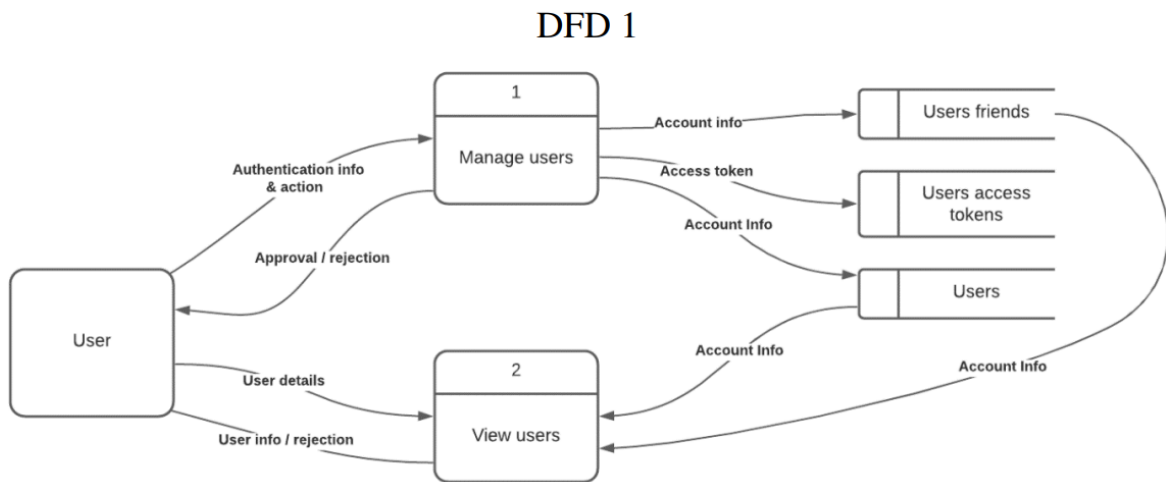


Рисунок 4.5 — Діаграма потоків даних. Процедура обробки даних користувача.

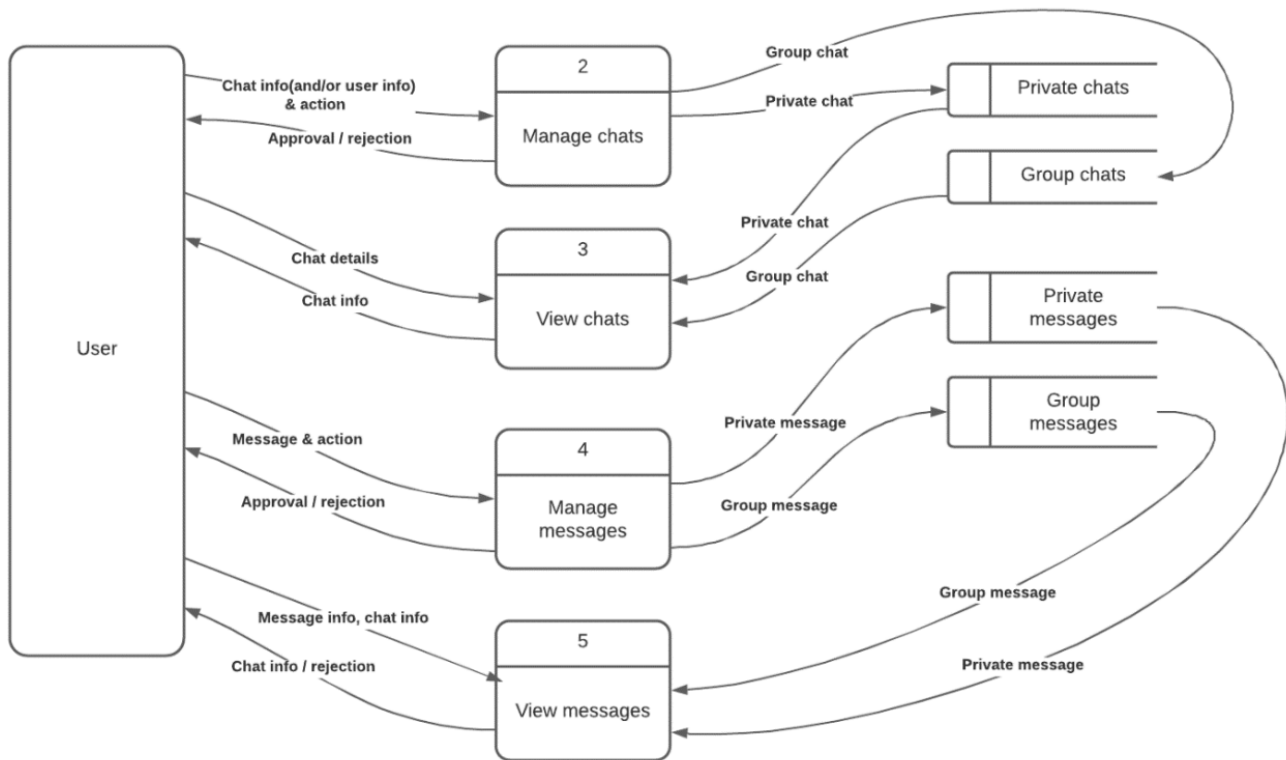


Рисунок 4.6 — Діаграма потоків даних. Процедура обробки даних чатів.

Як бачимо, користувач створює профіль та входить до системи, після чого сервіси починають його обробку. Для створення чату клієнт має додати до списку друзів інших користувачів та дочекатися підтвердження запиту. Після чого, користувач може створювати як приватний, так і груповий чати з іншими клієнтами, що входять до його списку друзів. Також, клієнт може отримувати запити на додавання до списку друзів від інших користувачів, та підтверджувати або відхиляти такі запити.

4.3 Інфологічна модель. Діаграма сутностей.

Діаграма сутності – це спеціалізована графіка, яка ілюструє взаємозв'язки між сутностями в базі даних. Діаграми ER використовують символи, що представляють три типи інформації: сутності (або поняття), відносини та атрибути.

Основними поняттями діаграм сутностей є:

- Сутність – це клас однотипних об'єктів, інформація про які повинна бути врахована в моделі;
- Екземпляр сутності – це конкретний представник даної сутності;
- Ключ сутності – це набір атрибутів, значення яких у сукупності є унікальними для кожного екземпляра сутності;
- Зв'язок – це відношення однієї сутності до іншої чи до самої себе. Можливо по одній сутності знаходити інші, зв'язані з нею.

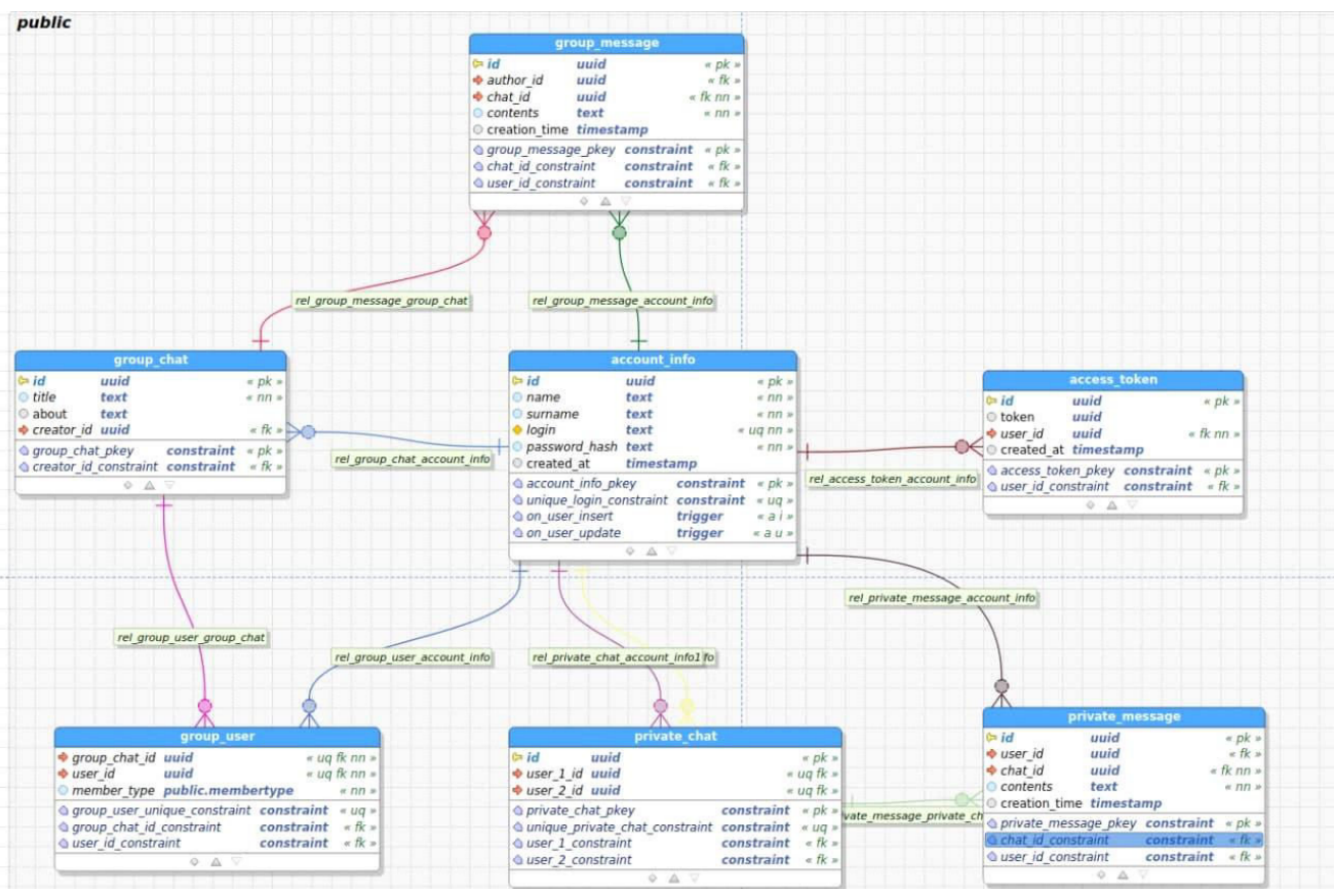


Рисунок 4.7 — Діаграма сутність – зв'язок.

В даній програмі, ми маємо 8 сутностей: користувач, ключ доступу до системи (Access Token), приватний чат, приватне повідомлення, груповий чат, групове повідомлення та сутність, що пов'язує користувача з груповим чатом (group_user). Користувач реєструється в системі, після чого сервіс опрацьовує та зберігає його дані. Далі користувач отримує ключ доступу до системи, який генерується сервісом, на основі свого логіну та паролю.

Користувач може створювати запити на додавання до списку друзів інших користувачів, які обробляються сервісом та зберігаються до таблиці “friends” зі статусом “Очікує підтвердження” (pending request). Після чого, якщо інший користувач повинен підтвердити запит на додавання до друзів, його статус запиту змінюється на “підтверджено” (request accepted).

Надалі користувач має змогу створювати чати з користувачами, що входять до списку його друзів. Ці запити оброблює сервіс та створює записи у відповідних таблицях: group_chat та group_user у разі створення групового чату, та private_chat при створенні приватного чату відповідно. Користувач має змогу створювати необмежену кількість групових чатів з одними і тими же користувачами, але може створювати лише один приватний чат з юзером, за умови, що такий чат не було створено відповідним користувачем раніше. Якщо користувач спробує створити чат з користувачем, з яким він вже має приватну кімнату, сервіс відповість помилкою. Проте, така можливість була заборонена на рівні клієнтського застосунку, перевіркою вже існуючих чатів з користувачами та формування списку на доступні чати після перевірки, що дає змогу обирати користувача, з яким буде створено приватний чат, лише зі списку друзів з яким поточний клієнт ще не має відповідного чату.

4.4 Нормалізація моделі.

В усіх таблицях значення у кожній клітинці є одиничними, усі записи у стовпчику мають один тип, кожний стовпчик має індивідуальне ім'я, порядок розташування рядків та стовпчиків є несуттєвим та відсутні повні повтори рядків, а отже наші відношення знаходяться у першій нормальній формі.

Окрім того, первинним ключем у кожній таблиці є одне єдине значення, тому відношення знаходяться у другій нормальній формі.

Також, будь-яке поле, що не залежить від основного ключа та від будь-якого іншого поля, винесене до окремої таблиці, а отже наші відношення знаходяться у третій нормальній формі.

На мою думку, такого рівня нормалізації моделі має бути достатньо для успішної реалізації поставлених задач.

4.5 Структура програми.

Враховуючи специфіку поставленої задачі, при розробці алгоритму використано метод покрокової деталізації, згідно з якою складний алгоритм розбивається на підзадачі, кожна з яких у свою чергу також розбивається на декілька простіших. Це дозволяє також розбити алгоритм на окремі модулі, у яких реалізовуватимуться окремі функції для роботи з даними та їх обробки. Враховуючи специфіку продукту, для забезпечення зручної роботи користувача, значна увага при розробці приділена організації графічного інтерфейсу програмного додатка та його функціональним можливостям.

Для зручної організації даних та доступу до них було вирішено використовувати об'єктно орієнтовану модель програмування.

Для проектування додатку використовувалась "Onion Architecture", а саме: в центрі знаходяться усі класи сутностей, у яких у нас зберігаються дані з бази даних (надалі БД), над ними добудовується шар репозиторіїв спілкування з БД, над ними

побудовані сервіси з реалізацією усіх необхідних методів, над ним надбудовано шар контролерів і на поверхні знаходяться безпосередньо інтерфейси взаємодії для користувачів. У репозиторіях було використано як вбудовані методи CRUD, так і власноруч створені запити для отримання звітів з використанням декількох таблиць або ж отримання агрегованих значень стовпчиків. Кожен контролер викликає відповідні методи сервісів, а сервіси взаємодіють з репозиторіями для отримання необхідної інформації або ж запису інформації до таблиці БД.

Для реалізації входу у систему та реєстрації було використано поле Authorization протоколу HTTP, у якому користувач передає свій ключ доступу до свого профілю та ресурсів, до яких він має доступ. При вході до системи, користувач надсилає JSON об'єкт з полями логіну та пароллю, на що він отримує ключ доступу, у разі, коли користувача було ідентифіковано. Також використовуються стандартні методи Spring Security, які постачаються фреймворком. Користувач має доступ до сервісу через мобільний додаток, який надсилає запити через протокол HTTP на доступ до відповідних ресурсів, які обробляються засобами Spring MVC. Також, було виконане оформлення сторінок в мобільному додатку за допомогою Android SDK, HTTP запити за допомогою бібліотеки Retrofit та WebSocket запити для реалізації отримання та відправки повідомлень у реальному часі за допомогою бібліотеки OkHTTP.

4.6 Реалізація взаємодії з базою даних.

Для створення схеми бази даних було створено файл опису таблиць `schema.sql`. Для взаємодії та підключення до бази даних використовувались можливості фреймворку Spring Boot. Так, для конфігурації підключення до потрібної бази даних було використано файл `application.properties`.

Для збереження даних з таблиць було створено відповідні класи та позначено їх анотаціями `@Entity`, `@Table(name = "назва відповідної таблиці в базі даних")`. Також були вказані первинні ключі, які було позначено анотаціями `@Id`,

@GeneratedValue(strategy = GenerationType.AUTO) для автоматичної генерації ключів базою даних, зв'язки між таблицями за допомогою анотацій @OneToOne, @OneToMany, @ManyToOne, @ManyToMany, створено класи для композитних ключів та позначено їх у класі сутності як @EmbeddedId.

Для безпосередньої взаємодії з даними БД було використано репозиторії. Окрім звичайних наданих репозиторіями методів також використовувалися власноруч створені запити за допомогою анотації @Query.

Також, для більш складних запитів було використано інтерфейс взаємодії EntityManager, за допомогою якого можна виконати запит мовою цільової СКБД та отримати результат запиту у вигляді масиву таблиць ключ - значення, які потім будуть вручну переведені до відповідних об'єктів.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для роботи програмного комплексу треба запустити серверний додаток, який буде працювати на порті 8080. Також треба запустити СКБД PostgreSQL на порті 5432.

Після чого він побачить наступний екран:

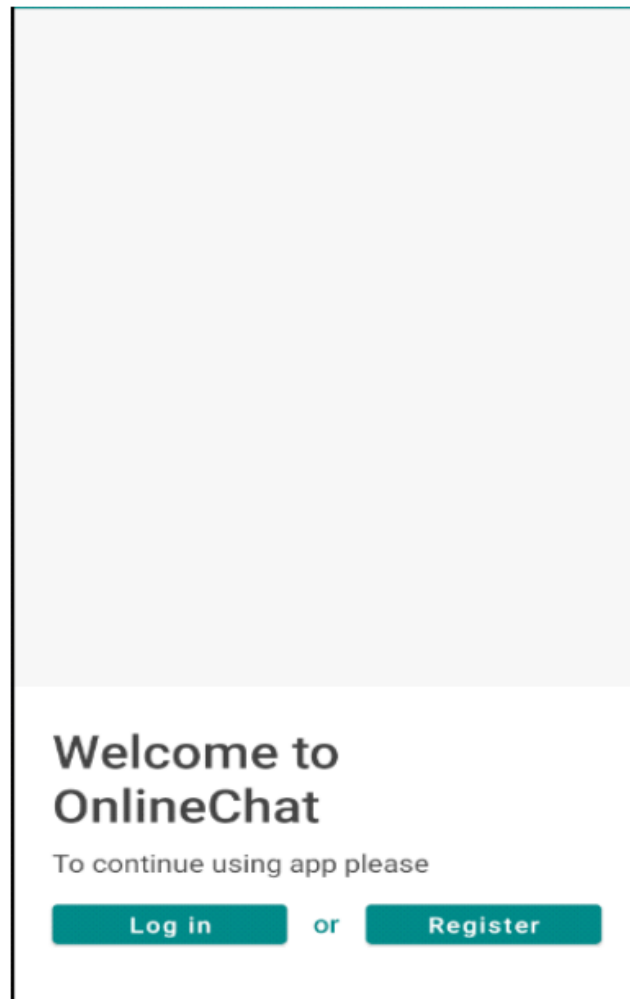


Рисунок 5.1 — Початковий екран.

На цьому екрані доступно дві опції: перейти до екрану реєстрації або увійти до екрану входу до системи. Щоб зареєструватися користувач має пройти на екран реєстрації, ввести дані до усіх полів та натиснути кнопку Create account.



Register

Login

First Name

Last Name

Password

Create account

Already have an account? [Log in](#)

Рисунок 5.2 — Екран реєстрації.

Після успішної реєстрації користувача перемістить до екрану входу до системи.



Login

Login

Password

Log in

Don't have an account?
Consider creating a new one

Register

Рисунок 5.3 — Екран входу до системи.

Після введення даних користувач має натиснути кнопку Log in для входу у систему, після чого йому доступний вибір екранів через бокове меню:

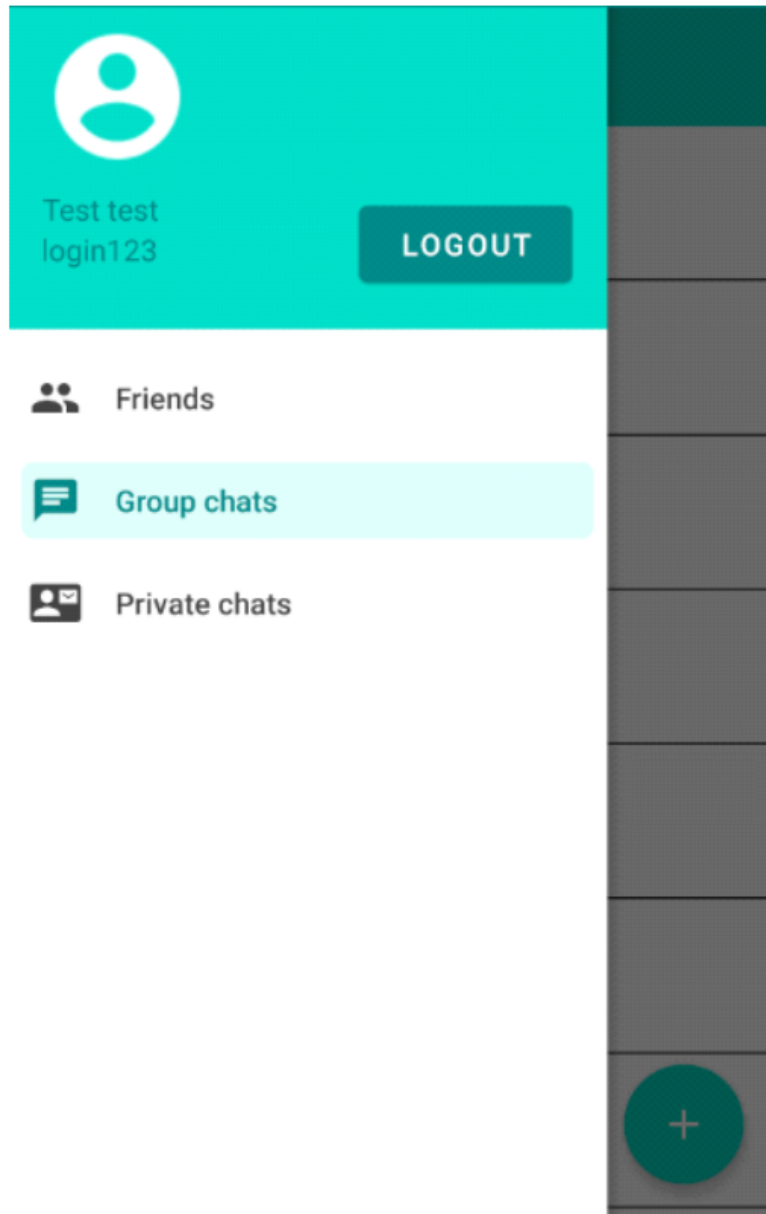


Рисунок 5.4 — Меню вибору.

Для додавання друзів користувач має натиснути кнопку “+” у правій нижній частині екрану. Після натискання цієї кнопки користувача перемістить до сторінки пошуку користувачів, яких ще немає у списку його друзів. На цьому екрані можна шукати користувачів за логіном та надсилати їм запити на додавання до списку друзів.

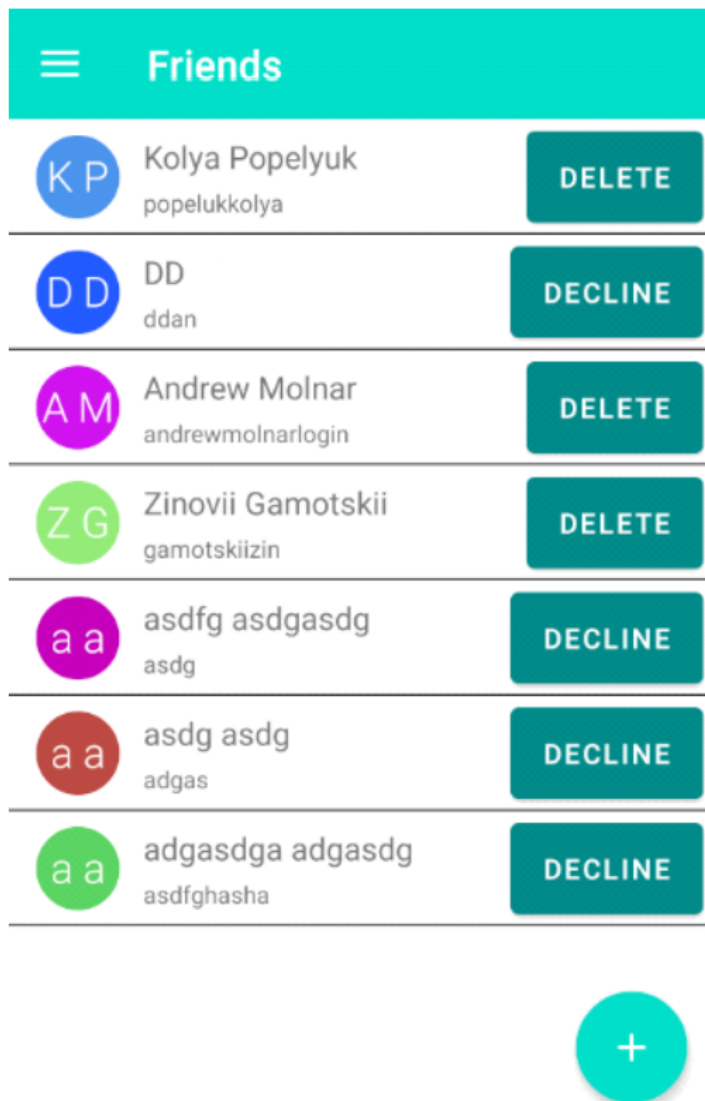


Рисунок 5.5 — Экран списка друзів.

Щоб створити новий груповий чат користувач так само має натиснути кнопку “+”. Після цього його перемістить на екран створення нового групового чату.

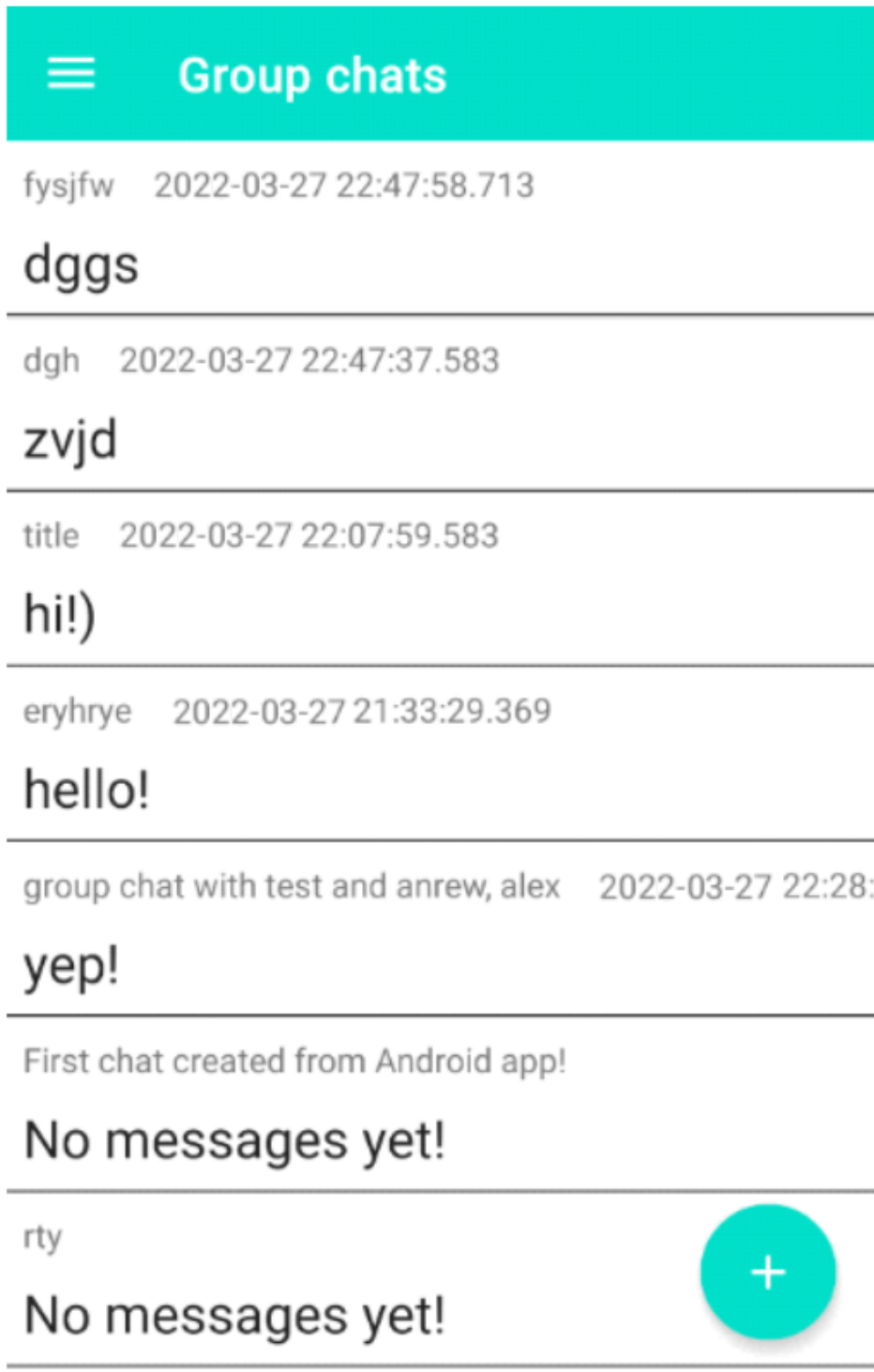


Рисунок 5.6 — Екран групових чатів.

← Create group chat

CREATE

title:

title

about:

about

participants:








	Kolya Popelyuk popelukkolya	<input checked="" type="checkbox"/>
	Andrew Molnar andrewmolnarlogin	<input checked="" type="checkbox"/>
	Zinovii Gamotskii gamotskiizin	<input checked="" type="checkbox"/>
	asdg asdg adgas	<input type="checkbox"/>
	DD ddan	<input type="checkbox"/>
	asdfg asdgasdg asdg	<input type="checkbox"/>
	adgasdga adgasdg asdfghasha	<input type="checkbox"/>

Рисунок 5.7 — Экран створення групового чату.

Після введення необхідних даних та вибору учасників чату користувач має натиснути кнопку Create щоб створити чат.

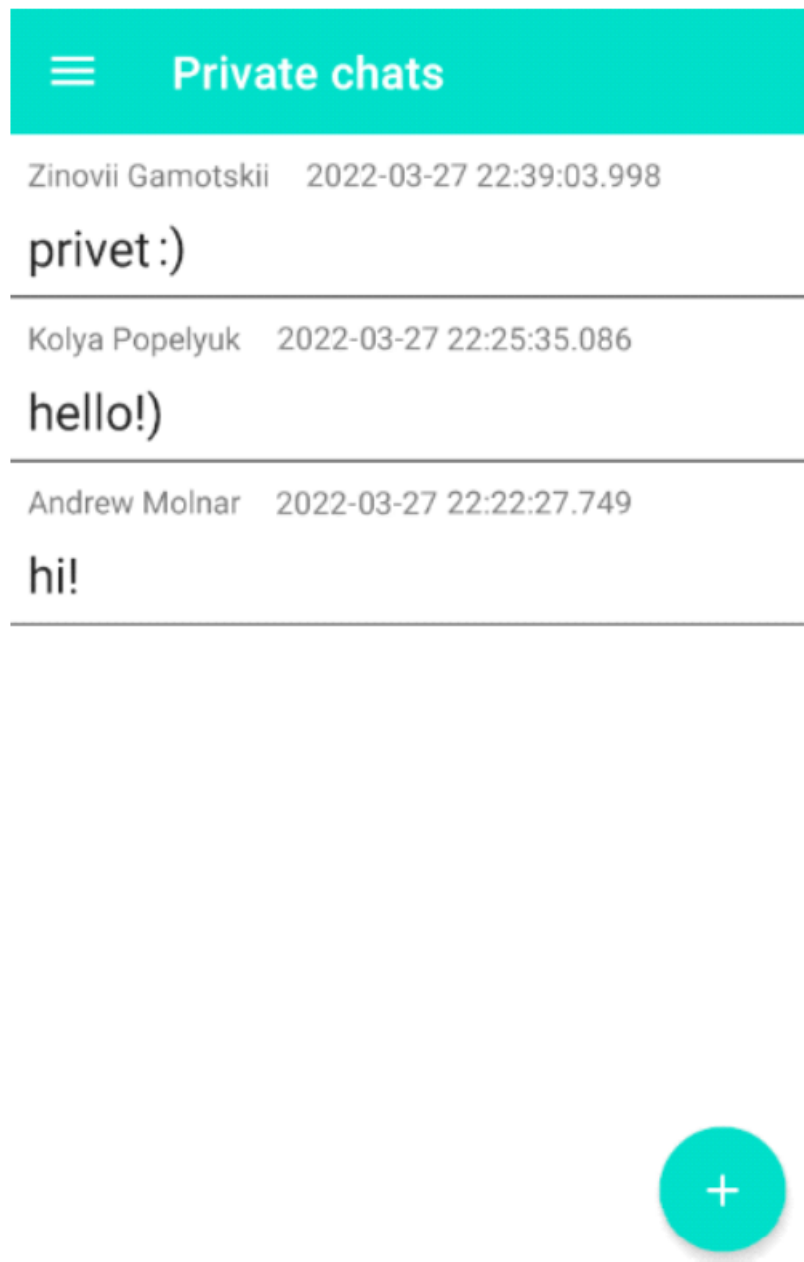


Рисунок 5.8 — Екран приватних чатів.

Створення приватного чату відбувається за попередньою схемою.

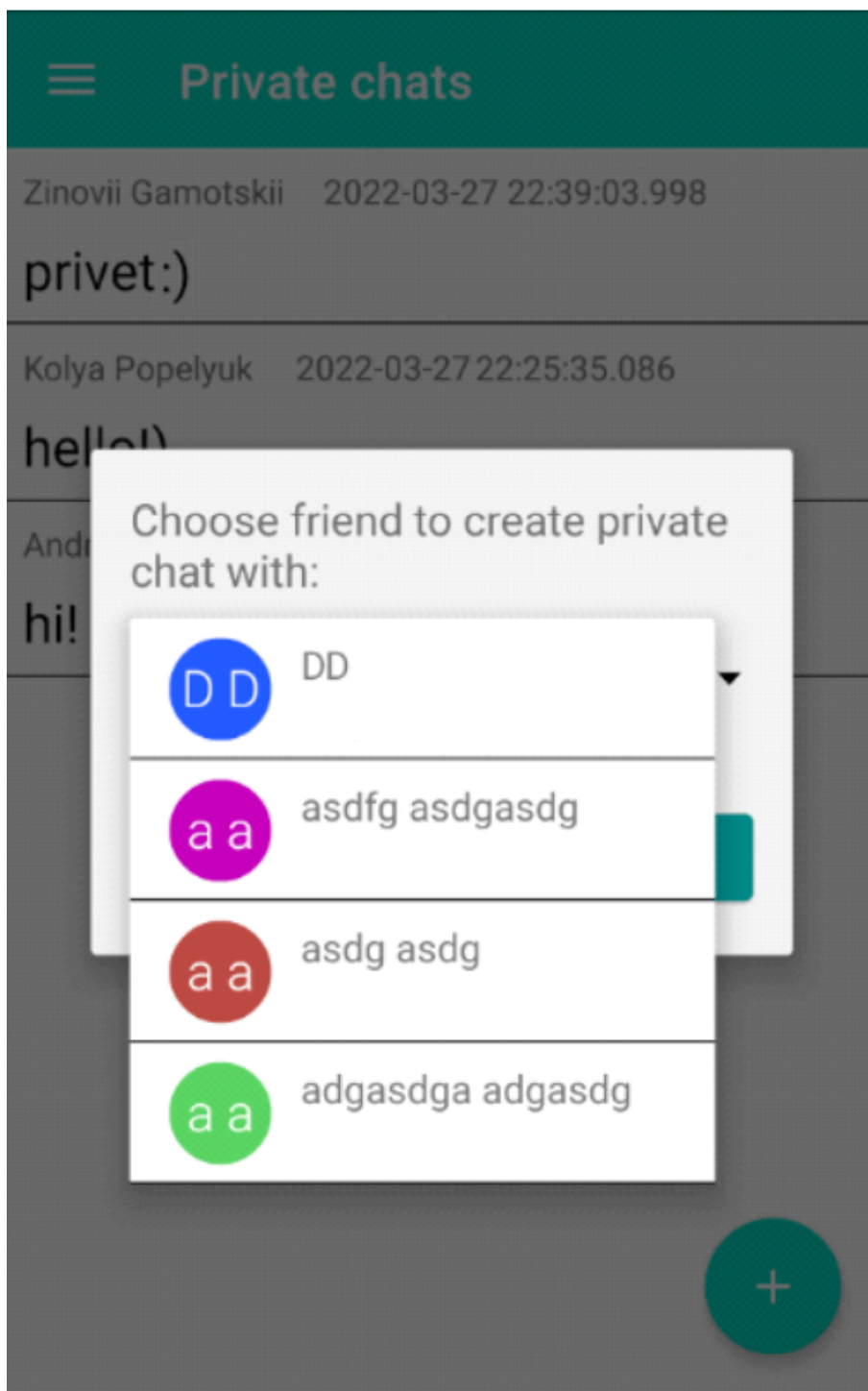


Рисунок 5.9 — Экран створення приватного чату.

Далі кожному з учасників групи буде відправлений запит на додавання в груповий чат, і при наступному запуску та підключенню до серверу, додаток автоматично додасть користувача та завантажить історію повідомлень.

ВИСНОВКИ

У ході виконання даної роботи було розроблено програму для обміну повідомленнями. Найвагомішою частиною була розробка структури БД, відношень між таблицями, проектування таблиць відповідно до нормальних форм, а також правильна побудова запитів.

Протягом роботи були отримані основні необхідні при розробці баз даних навички, а саме:

- Створення концептуальної моделі бази даних: діаграми потоків даних;
- Створення інфологічної моделі бази даних: діаграми “сутність– зв’язок”;
- Створення складних запитів мовою SQL, використання з’єднань таблиць, нормалізації.

Оскільки робота є практичною, то в процесі розробки були здобуті навички роботи з різними технологіями:

- Spring Boot фреймворк мови програмування Java, що використовувався у якості бекенду;
- Android SDK фреймворк для створення мобільних додатків у парі з мовою програмування Kotlin;
- Отримано навички у під’єднанні БД до бекенду та реалізації взаємодії бекенду з БД за допомогою бібліотек Spring Boot.

Оглянувши проект після завершення його розробки, можна виділити його плюси:

- Система авторизації;
- Перспективність та актуальність;
- Використання сучасних технологій;
- Максимально простий інтерфейс;

Проте також є недоліки:

- Дещо спрощена бізнес-логіка;
- Відсутність пагінації;
- Занадто спрощений інтерфейс без достатнього оформлення, відсутність стилізації.

Усі ці недоліки дуже можна вирішити, враховуючи усі отримані у процесі розробки навички, якщо продовжувати підтримувати проект і надалі. Можна значно розширити та покращити бізнес-логіку та створити більш якісний інтерфейс.

Результат цієї роботи, на мою думку, можна використовувати як кінцевий продукт після деяких правок , покращень та контейнеризації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. stackoverflow.com [Електронний ресурс] — Режим доступу до ресурсу: <https://stackoverflow.com/>
2. Молінаро Е. SQL. Збірка рецептів. Переклад з англ.: Символ!Плюс, 2009. 672 с.
3. Теорія та практика побудов баз даних. 8-е видання. / Д. Крйонке. 2003. 800 с.
4. Шварц Б., Зайцев П., Ткаченко В. та інш. MySQL. Оптимізація продуктивності (2-е видання) 2010.pdf
5. Documentation. Manuals. The PostgreSQL Global Development Group. Режим доступу до ресурсу: <https://www.postgresql.org/docs/>
6. Hibernate ORM. Documentation-5.4. Hibernate. Режим доступу до ресурсу: <https://hibernate.org/orm/documentation/5.4/>
7. Spring Framework Documentation. Version 5.3.2. Spring. Last updated 2020-12-09. URL: <https://docs.spring.io/springframework/docs/current/reference/html/>
8. Вільна енциклопедія Вікіпедія [Електронний ресурс] : IntelliJ IDEA - Квітень 2022, 28 — Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/IntelliJ_IDEA
9. Офіційний сайт Kotlin Language [Електронний ресурс] : FAQ - Січень 2022, 11. — Режим доступу до ресурсу: <https://kotlinlang.org/docs/faq.html>
10. Офіційний сайт Postman [Електронний ресурс] : Postman Company — Січень 2022, 25. — Режим доступу до ресурсу: <https://www.getpostman.com/company>
11. Куліков С.С. Тестування програмного забезпечення. Базовий курс: практичний додаток — Мінськ: видавництво “Чотири четвірки”, 2015 — 294 с.
12. Вільна енциклопедія Вікіпедія [Електронний ресурс] : Telegram — Березень 2022, 14 — Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Telegram>
13. Вільна енциклопедія Вікіпедія [Електронний ресурс] : Viber — Березень 2022, 16 — Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/Viber>

14. Вільна енциклопедія Вікіпедія [Електронний ресурс] : WhatsApp — Березень 2022, 18 — Режим доступу до ресурсу: <https://uk.wikipedia.org/wiki/WhatsApp>.
15. Вільна енциклопедія Вікіпедія [Електронний ресурс] : Клієнт—серверна архітектура — Березень 2022, 20 — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Client%E2%80%93server_model.
16. Leo S. Hsu, Regina O. Obe. PostgreSQL: Up and Running. /O'Reilly Media, 2012. — P. 167.
17. Hans-Jurgen Schonig. Troubleshooting PostgreSQL // Packt Publishing, 2015. — P. 164.
18. Pierre-Yves Saumont. The Joy of Kotlin // Simon and Schuster, 2019. — P. 802.
19. Вільна енциклопедія Вікіпедія [Електронний ресурс]: DFD — Травень 2022, 02— Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Data-flow_diagram
20. ДСТУ 8302:2015 “Інформація та документація. Бібліографічне посилання. Загальні вимоги та правила складання”

ДОДАТОК А

Месенджер в режимі реального часу для он-лайн курсу “Структури даних”

Текст програмного модулю

УКР.НТУУ “КПІ ім. Ігоря Сікорського” ТЕФ_АПЕПС_ТР81____22Б 12-1

Аркушів 14

2020

```

//////////AccessTokenDTO
package com.BSS.onlinechat.api.dto

import com.google.gson.annotations.SerializedName
import java.security.InvalidParameterException
import java.sql.Timestamp

data class AccessTokenDTO(
    @SerializedName("access_token")
    var accessToken: String? = null,

    @SerializedName("created_at")
    var createdAt: Timestamp? = null
) {
    fun mapToAccessToken(): String =
        accessToken ?: throw InvalidParameterException("Access token is
null")
}

//////////AccountRetrievalDTO
package com.BSS.onlinechat.api.dto

import com.BSS.onlinechat.api.model.AccountInfo

data class AccountInfoRetrievalDTO(
    var id: String? = null,
    var name: String? = null,
    var surname: String? = null
) {
    fun mapToAccountInfo() =
        AccountInfo(id!!, name!!, surname!!)

    fun getInitials(): String {
        val nameLetter = name ?: ""
        val surnameLetter = surname ?: ""
    }
}

```

```

var initials = ""
if (nameLetter.isNotBlank()) {
    initials += nameLetter[0] + " "
}

if (surnameLetter.isNotBlank()) {
    initials += surnameLetter[0]
}

return initials
}

}

//////////GroupChatRetrievalDTO
package com.BSS.onlinechat.api.dto

import com.google.gson.annotations.SerializedName
import java.util.*

data class GroupChatRetrievalDTO(
    var id: UUID? = null,
    var title: String? = null,
    var about: String? = null,
    var creator: AccountInfoRetrievalDTO? = null,
    var participants: List<AccountInfoRetrievalDTO>? = null,
    @SerializedName("last_message")
    var lastMessage: GroupMessageRetrievalDTO? = null
)

package com.BSS.onlinechat.api.dto;

```

```

import com.google.gson.annotations.SerializedName

data class GroupMessageCreationDTO(
    private val contents: String,

    @SerializedName("chat_id")
    private val chatId: String
)

//////////GroupMessageRetrievalDTO
package com.BSS.onlinechat.api.dto

import com.google.gson.annotations.SerializedName
import java.sql.Timestamp
import java.util.*

data class GroupMessageRetrievalDTO(
    var id: String? = null,
    var author: AccountInfoRetrievalDTO? = null,

    @SerializedName("chat_id")
    var chatId: UUID? = null,
    var contents: String? = null,

    @SerializedName("created_at") val createdAt: Timestamp? = null
)

//////////LoginInfoDTO
package com.BSS.onlinechat.api.dto

data class LoginInfoDTO(val login: String, val password: String)

```

```

//////////AccountInfo
package com.BSS.onlinechat.api.model

data class AccountInfo(
    var id: String,
    val name: String,
    val surname: String
)

//////////RestWrapper
package com.BSS.onlinechat.api.rest.rest_wrapper

import com.BSS.onlinechat.api.dto.LoginInfoDTO
import com.BSS.onlinechat.api.model.AccountInfo
import com.BSS.onlinechat.api.rest.ApiEndpoints

interface RestWrapper {
    suspend fun login(loginCredentials: LoginInfoDTO)
    fun getApi(): ApiEndpoints
    fun getMyAccount(): AccountInfo
    fun getAccessToken(): String
    suspend fun saveAccessToken(accessToken: String)
    fun saveMyAccount(accountInfo: AccountInfo)
}

//////////RestWrapperImpl
package com.BSS.onlinechat.api.rest.rest_wrapper

import com.BSS.onlinechat.api.dto.LoginInfoDTO
import com.BSS.onlinechat.api.model.AccountInfo
import com.BSS.onlinechat.api.rest.ApiEndpoints
import
com.BSS.onlinechat.util.shared_preferences.SharedPreferencesWrapper
import com.google.gson.GsonBuilder
import kotlinx.coroutines.Dispatchers

```

```

import kotlinx.coroutines.withContext
import okhttp3.Interceptor
import okhttp3.OkHttpClient
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import javax.inject.Inject

private const val BASE_URL = "http://192.168.0.11:8080/api/"
private const val TAG = "Repository"

class RestWrapperImpl
@Inject constructor(
    private val sharedPreferencesWrapper: SharedPreferencesWrapper
) : RestWrapper {
    private val retrofit: Retrofit
    private val api: ApiEndpoints

    private var accessToken: String = ""
    private lateinit var myAccount: AccountInfo

    init {
        val gson = GsonBuilder().create()

        val httpClient = OkHttpClient
            .Builder()
            .addInterceptor(AuthInterceptor())
            .build()

        retrofit = Retrofit
            .Builder()
            .client(httpClient)
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create(gson))

```

```

        .build()

api = retrofit.create(ApiEndpoints::class.java)
}

override fun getApi(): ApiEndpoints {
    if (accessToken.isEmpty()) {
        throw IllegalStateException("Api consuming without logging")
    }
    return api
}

override suspend fun login(loginCredentials: LoginInfoDTO) =
withContext(Dispatchers.IO) {

saveAccessToken(api.login(loginCredentials).mapToAccessToken())
    saveMyAccount(api.getMyAccount().mapToAccountInfo())
}

override fun getAccessToken(): String = accessToken

override suspend fun saveAccessToken(accessToken: String) {
    this.accessToken = accessToken
    sharedPreferencesWrapper.saveAccessToken(accessToken)
}

override fun saveMyAccount(accountInfo: AccountInfo) {
    this.myAccount = accountInfo
}

override fun getMyAccount(): AccountInfo = myAccount

inner class AuthInterceptor : Interceptor {
    override fun intercept(chain: Interceptor.Chain):
okhttp3.Response {

```

```

        val newRequestBuilder = chain.request().newBuilder()

        if (accessToken.isNotEmpty()) {
            newRequestBuilder.addHeader("Authorization", "Bearer
$accessToken")
        }

        val newRequest = newRequestBuilder.build()
        return chain.proceed(newRequest)
    }
}

package com.BSS.onlinechat.api.rest

import com.BSS.onlinechat.api.dto.*
import retrofit2.http.*

private const val USER_SERVICE = "user-service/auth"
private const val GROUP_CHAT_SERVICE = "group-chat-service/chats"
interface ApiEndpoints {
    @PUT("$USER_SERVICE/login")
    suspend fun login(@Body loginCredentials: LoginInfoDTO):
AccessTokenDTO

    @GET("$USER_SERVICE/account")
    suspend fun getMyAccount(): AccountInfoRetrievalDTO

    @GET("$GROUP_CHAT_SERVICE/all")
    suspend fun getAllGroupChats(): List<GroupChatRetrievalDTO>

    @GET("$GROUP_CHAT_SERVICE/{chat_id}/messages")
    suspend fun getAllMessagesForGroupChat(@Path("chat_id") chatId:
String): List<GroupMessageRetrievalDTO>

```

```

@GET("$GROUP_CHAT_SERVIVCE/all")
suspend fun getAllPrivateChats(): List<GroupChatRetrievalDTO>

}

package com.BSS.onlinechat.api.ws

import com.google.gson.Gson
import com.google.gson.annotations.SerializedName
import com.google.gson.reflect.TypeToken
import com.google.gson.stream.JsonReader
import java.io.StringReader

sealed class WebSocketEvent {
    object ConnectionOpening : WebSocketEvent()
    object ConnectionClosing : WebSocketEvent()
    object ConnectionOpened : WebSocketEvent()
    object ConnectionClosed : WebSocketEvent()
    object ConnectionFailure : WebSocketEvent()
    object ReconnectionAttempt : WebSocketEvent()

    data class WebSocketMessage<T>(
        var payload: T? = null,

        @SerializedName("message_type")
        var messageType: WebSocketMessageType? = null
    ) : WebSocketEvent()
}

inline fun <reified T> websocketMessageFromJson(json: String):
WebSocketEvent.WebSocketMessage<T> {
    val reader = JsonReader(StringReader(json)).apply {
        isLenient = true
    }
}

```

```

return Gson().fromJson(
    reader,
    TypeToken.getParameterized(
        WebSocketEvent.WebSocketMessage::class.java,
        T::class.java
    ).type
)
}

```

```

inline fun <reified T> websocketMessageToJson(message:
WebSocketEvent.WebSocketMessage<T>): String =
    Gson().toJson(
        message,
        TypeToken.getParameterized(
            WebSocketEvent.WebSocketMessage::class.java,
            T::class.java
        ).type
    )

```

```
package com.BSS.onlinechat.api.ws
```

```

import com.google.gson.Gson
import com.google.gson.annotations.SerializedName
import com.google.gson.reflect.TypeToken
import com.google.gson.stream.JsonReader
import java.io.StringReader

```

```

sealed class WebSocketEvent {
    object ConnectionOpening : WebSocketEvent()
    object ConnectionClosing : WebSocketEvent()
    object ConnectionOpened : WebSocketEvent()
    object ConnectionClosed : WebSocketEvent()
    object ConnectionFailure : WebSocketEvent()
    object ReconnectionAttempt : WebSocketEvent()

    data class WebSocketMessage<T>(

```

```

var payload: T? = null,

@SerializedName("message_type")
var messageType: WebSocketMessageType? = null
) : WebSocketEvent()
}

inline fun <reified T> websocketMessageFromJson(json: String):
WebSocketEvent.WebSocketMessage<T> {
    val reader = JsonReader(StringReader(json)).apply {
        isLenient = true
    }

    return Gson().fromJson(
        reader,
        TypeToken.getParameterized(
            WebSocketEvent.WebSocketMessage::class.java,
            T::class.java
        ).type
    )
}

inline fun <reified T> websocketMessageToJson(message:
WebSocketEvent.WebSocketMessage<T>): String =
    Gson().toJson(
        message,
        TypeToken.getParameterized(
            WebSocketEvent.WebSocketMessage::class.java,
            T::class.java
        ).type
    )

package com.BSS.onlinechat.api.ws

enum class WebSocketMessageType {

```

```

        GROUP_MESSAGE_PATCH, GROUP_MESSAGE_CREATE, GROUP_MESSAGE_DELETE,
        PRIVATE_MESSAGE_PATCH, PRIVATE_MESSAGE_CREATE,
PRIVATE_MESSAGE_DELETE
    }

```

```

package com.BSS.onlinechat.api.ws

```

```

import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import com.BSS.onlinechat.api.rest.rest_wrapper.RestWrapper
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.asCoroutineDispatcher
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch
import okhttp3.*
import okio.ByteString
import java.util.concurrent.Executors
import javax.inject.Inject

```

```

private const val TAG = "WebSocketWrapper"

```

```

private const val NUM_OF_THREADS = 1
private const val REMOTE_HOST_URL = "ws://192.168.0.11:8080/api/ws"
private const val AUTHORIZATION_HEADER = "Authorization"
private const val BEARER_PREFIX = "Bearer "

```

```

class WebSocketWrapper @Inject constructor(
    private val restWrapper: RestWrapper,
    private val client: OkHttpClient
) {
    private lateinit var websocket: WebSocket

    private val websocketListener: WebSocketListener =
ChatWebSocketListener()

```

```

    private val dispatcher =
Executors.newFixedThreadPool(NUM_OF_THREADS).asCoroutineDispatcher()

    private val _eventBus: MutableLiveData<WebSocketEvent> =
MutableLiveData()
    val eventBus: LiveData<WebSocketEvent>
get() = _eventBus

    fun send(message: String) {
Log.d(TAG, "WebSocketWrapper: sending $message")
websocket.send(ByteString.encodeString(message, Charsets.UTF_8))
    }

    fun connect() {
if (this::websocket.isInitialized) {
    websocket.cancel()
}

val requestBuilder: Request.Builder = Request
    .Builder()
    .url(REMOTE_HOST_URL)
    .addHeader(
        AUTHORIZATION_HEADER,
        BEARER_PREFIX + restWrapper.getAccessToken()
    )

    websocket = client.newWebSocket(requestBuilder.build(),
websocketListener)
    }

    private fun reconnect() {
websocket.cancel()
_eventBus.postValue(WebSocketEvent.ReconnectionAttempt)

```

```

Log.d(TAG, "reconnection attempt")
CoroutineScope(dispatcher).launch {
    delay(5000)
    connect()
}
}

inner class ChatWebSocketListener : WebSocketListener() {

override fun onOpen(webSocket: WebSocket, response: Response) {
    Log.i(TAG, "onOpen: $webSocket")
    _eventBus.postValue(WebSocketEvent.ConnectionOpening)
}

override fun onFailure(webSocket: WebSocket, t: Throwable,
response: Response?) {
    Log.e(TAG, "onFailure: ${t.message} ${response.toString()}")
    reconnect()
    _eventBus.postValue(WebSocketEvent.ConnectionFailure)
}

override fun onMessage(webSocket: WebSocket, bytes: ByteString) {
    try {
        val obj =
websocketMessageFromJson<Object>(bytes.string(Charsets.UTF_8))
        Log.i(TAG, "onMessage: $obj")
        _eventBus.postValue(obj)
    } catch (e: Exception) {
        Log.d(TAG, "onMessage ERROR: ${e.message}")
    }
}

override fun onClose(webSocket: WebSocket, code: Int, reason:
String) {
    Log.i(TAG, "onClosed: ")
}
}

```

```
        _eventBus.postValue(WebSocketEvent.ConnectionClosed)
    }
}
}
```