Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
в.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: Програмний застосунок забезпечення конфіденційного обміну голосовими повідомленнями

Виконавець: студент IV курсу, групи КБ-42

_____ **Гліб Кириченко**
(підпис) (ім'я, прізвище)

	Ім'я, прізвище	**Підпис**
Керівник	Андрій ФЕСЕНКО	

Нормоконтроль	Андрій БІГДАН	

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
в.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студента **КБ-42** **Кириченка Гліба Андрійовича**
(група) (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Програмний застосунок забезпечення конфіденційного обміну голосовими повідомленнями

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №3 від 20.10.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Забезпечення конфіденційності голосових повідомлень у месенджерах.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно провести огляд месенджерів-аналогів щодо захисту голосових повідомлень, сформувати стек майбутнього програмного застосунку, розробити застосунок, проаналізувати існуючі методи шифрування та впровадити покращений метод забезпечення конфіденційності для голосових повідомлень.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність полягає у створенні програмного застосунку для конфіденційного обміну голосовими повідомленнями.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 24 жовтня 2022 року

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ім'я, прізвище)

Завдання прийняла до виконання _____ (підпис) Гліб КИРИЧЕНКО (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	21.10.2022 – 22.01.2023	*виконано*
2	Аналіз літератури	29.01.2023 – 11.02.2023	*виконано*
3	Обґрунтування вибору рішення	12.02.2023 – 15.02.2023	*виконано*
4	Огляд месенджерів-аналогів	16.02.2023 – 04.03.2023	*виконано*
5	Вибір стеку для майбутнього програмного застосунку	05.03.2023 – 21.03.2023	*виконано*
6	Розробка програмного застосунку	22.03.2023– 08.04.2023	*виконано*
7	Впровадження методу забезпечення конфіденційності голосових повідомлень	09.04.2023 – 10.05.2023	*виконано*
8	Оформлення пояснювальної записки	11.05.2023 – 27.05.2023	*виконано*
9	Підготовка до захисту кваліфікаційної роботи	28.05.2023 – 05.06.2023	*виконано*

Завдання видав _____ (підпис) Андрій ФЕСЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Гліб КИРИЧЕНКО (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний застосунок забезпечення конфіденційного обміну голосовими повідомленнями» складається зі вступу, основної частини, що містить 4 розділи, висновків і списку літератури та джерел. Основний обсяг роботи – 69 сторінки. Робота містить 23 рисунки, 1 таблицю та 3 додатки. Список використаних джерел включає 59 джерел.

Мета роботи - розробка програмного застосунку для конфіденційного обміну голосовими повідомленнями з використанням методів та алгоритмів, які забезпечують високий рівень безпеки та захисту даних.

Об’єкт дослідження – процес конфіденційного обміну голосовими повідомленнями в месенджерах.

Предмет дослідження – метод захищеного обміну голосовими повідомленнями, включаючи аналіз існуючих рішень, вимоги до програмного застосунку, проектування та розробку.

Метод дослідження - полягає в аналізі існуючих рішень, визначенні вимог та функціональності, проектуванні системи та реалізації програмного застосунку для конфіденційного обміну голосовими повідомленнями.

Практичне завдання роботи полягає у створенні програмного застосунку, який забезпечує конфіденційний обмін голосовими повідомленнями між користувачами та має високий рівень захисту даних.

Результати дослідження можуть бути використані для поліпшення технологій конфіденційного обміну даними та захисту приватності користувачів у мобільних додатках.

Для досягнення мети роботи передбачається розв'язання наступних завдань:

1. Аналіз існуючих методів шифрування в системах обміну повідомленнями.
2. Розробка вимог до програмного засобу для конфіденційного обміну голосовими повідомленнями.

3. Розробка архітектури програмного засобу.

4. Розробка програмного коду застосунку.

Напрямки подальших досліджень:

1. Вивчення та застосування нових методів шифрування та захисту даних для забезпечення більш високого рівня конфіденційності та безпеки.

2. Аналіз використання програмного застосунку користувачами та дослідження їх задоволеності функціональністю та якістю обслуговування.

3. Вивчення та застосування нових технологій управління базами даних та оптимізація їх використання у програмному застосунку.

4. Вивчення можливості інтеграції програмного застосунку з іншими додатками та платформами для забезпечення більш широкої функціональності та зручності користування.

Вимоги до програмного засобу для конфіденційного обміну голосовими повідомленнями:

1. Криптографічна безпека: Додаток повинен використовувати надійні алгоритми шифрування для забезпечення конфіденційності голосових повідомлень. Шифрування повинно бути сильним і відповідати сучасним стандартам безпеки.

2. Керування ключами: Додаток повинен мати механізм для безпечного обміну та керування ключами шифрування між користувачами. Ключі повинні бути збережені в безпечному місці.

3. Аутентифікація користувачів: Додаток повинен мати механізм аутентифікації користувачів для запобігання несанкціонованому доступу до голосових повідомлень.

4. Інтерфейс користувача: Додаток повинен мати зручний та інтуїтивно зрозумілий інтерфейс для забезпечення зручного використання голосового обміну повідомленнями.

Ключові слова: месенджер, конфіденційний обмін голосовими повідомленнями, блочний шифр AES, програмний застосунок, firebase, шифрування, архітектура, клієнтська частина, серверна частина, програмна реалізація.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

UI	–	User interface
IT	–	Information Technology
AES	–	Advanced Encryption Standart
VPN	–	Virtual Private Network
API	–	Application Programming Interface
FCM	–	Firebase Cloud Messaging
IDE	–	Integrated Development Environment
FRD	–	Firebase Realtime Database
RSA	–	Rivest–Shamir–Adleman
E2EE	–	End-to-end encryption
SDK	–	Software Development Kit
ПЗ	–	Програмне забезпечення
ІКТ	–	Інформаційно-комунікаційні технології

ЗМІСТ

ВСТУП

В сучасному світі інформаційні технології займають все більш вагому роль у нашому житті. Кожен день ми стикаємося з безліччю електронних засобів комунікації, які дозволяють нам легко та швидко спілкуватися з іншими людьми, незалежно від відстані та місця перебування [1]. Голосові повідомлення є одним із таких ефективних засобів комунікації, які забезпечують швидкий та зручний обмін інформацією між користувачами.

Однак, разом зі зручністю та швидкістю обміну, пов'язані й питання безпеки. Кожен з нас прагне зберегти приватність та конфіденційність своїх даних, тому захист інформації є надзвичайно важливим аспектом в розробці програмних засобів комунікації.

У зв'язку з цим, актуальним завданням є розробка програмного застосунку для конфіденційного обміну голосовими повідомленнями з використанням методів та алгоритмів, які забезпечують високий рівень безпеки та захисту даних.

У рамках даної роботи також будуть вивчені та проаналізовані сучасні методи та алгоритми шифрування та дешифрування даних, а також методи аутентифікації користувачів та захисту від атак типу "людина посередині" (man-in-the-middle).

Окрім теоретичної складової, у роботі буде реалізовано практичну частину - розробка програмного коду застосунку та його тестування. Також буде проведено порівняльний аналіз розробленого програмного засобу з існуючими аналогами на ринку.

Розроблений програмний засіб може бути використаний у компаніях, що працюють з конфіденційною інформацією, у банківському та медичному секторах, у сфері онлайн-торгівлі та електронної комерції, а також в освітніх та наукових закладах.

Результатом дослідження має стати розроблений програмний засіб, який буде забезпечувати конфіденційний обмін голосовими повідомленнями. Дипломна робота передбачає розробку вимог до програмного засобу, розробку архітектури,

програмного коду та проведення тестування. При цьому буде проведено порівняльний аналіз розробленого програмного засобу з існуючими аналогами на ринку.

РОЗДІЛ 1
ПОНЯТТЯ МЕССЕНДЖЕРУ

1.1 Основні поняття та визначення

За останні кілька років месенджери стали невід'ємною складовою сучасної комунікації. Месенджери - це додатки для мобільних та стаціонарних пристроїв, які дозволяють користувачам обмінюватися повідомленнями, зображеннями, відео та аудіофайлами у режимі реального часу [2]. Окрім цього, деякі месенджери також підтримують голосовий та відеозв'язок. Основна перевага месенджерів полягає в тому, що вони дозволяють користувачам спілкуватися в будь-який час та з будь-якого місця, використовуючи тільки доступ до Інтернету. У порівнянні з традиційними методами комунікації, такими як телефонні дзвінки та SMS, месенджери є більш економічним та зручним засобом комунікації.

Однак, разом з популярністю месенджерів зростає також і проблема конфіденційності даних, зокрема голосових повідомлень. Зазвичай, месенджери зберігають дані користувачів на своїх серверах, що може становити загрозу для їх конфіденційності. Також, у випадку, якщо користувачі обмінюються голосовими повідомленнями, ці дані можуть бути перехоплені зловмисниками, що також порушує їх конфіденційність [3]. У зв’язку з цим, розробка програмного застосунку для конфіденційного обміну голосовими повідомленнями є актуальною та необхідною задачею.

Основні функціональні можливості месенджерів включають в себе:

- Обмін текстовими повідомленнями
- Обмін зображеннями та відеофайлами
- Голосовий та відеозв'язок
- Створення групових чатів
- Використання емодзі та стікерів
- Підтримка ботів та інтеграція з різноманітними сервісами

1.2 Типи месенджерів

Залежно від способу функціонування, можна виділити два типи месенджерів – централізовані та децентралізовані (рисунок 1.1).

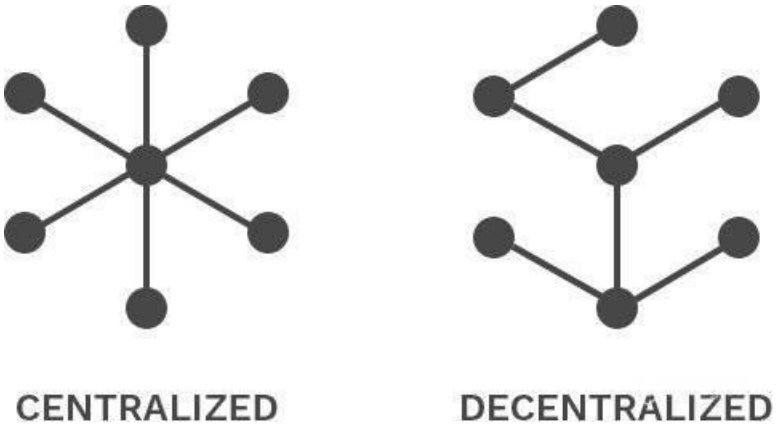

Рисунок 1.1 – Централізація та децентралізація

Централізовані месенджери: в цьому випадку всі дані передаються через центральний сервер, який забезпечує роботу месенджера та зберігання інформації. Централізовані месенджери зазвичай мають більш високий рівень захисту даних, оскільки управління безпекою здійснюється централізовано. Проте, такі месенджери можуть стати легкою мішенню для хакерів або зловмисників, оскільки один сервер містить велику кількість даних користувачів [4].

Децентралізовані месенджери: такі месенджери не використовують центральний сервер для передачі даних між користувачами. Замість цього, користувачі можуть взаємодіяти напряму між собою, за допомогою різних протоколів та мереж. Однією з переваг децентралізованих месенджерів є те, що вони мають більший рівень приватності, оскільки дані не зберігаються на центральному сервері. Однак, такі месенджери можуть бути менш надійними і менш ефективними у відправці повідомлень [5].

Крім того, можна виділити такі типи месенджерів за їх функціональністю:

1) Голосові месенджери. Такі месенджери дозволяють користувачам відправляти голосові повідомлення один одному. Це дуже зручно, коли користувач не може або не хоче писати текстові повідомлення.

2) Відео-месенджери. Такі месенджери дозволяють користувачам обмінюватися відео-повідомленнями один одному. Вони дозволяють бачити один одного в режимі реального часу та спілкуватися майже так, як при зустрічі обличчя в обличчя.

3) Текстові месенджери. Це найпоширеніший тип месенджерів, який дозволяє користувачам відправляти текстові повідомлення один одному. Вони є швидким та зручним засобом комунікації, що використовується мільйонами людей по всьому світу [6].

1.3 Аналіз найпопулярніших існуючих аналогів

Можна виділити кілька аналогів програмного забезпечення для конфіденційного обміну голосовими повідомленнями:

- Signal – це безкоштовний месенджер, який пропонує закриті групові чати, відео- та голосові дзвінки. Signal також пропонує шифрування повідомлень на рівні клієнта, що означає, що навіть розробники Signal не можуть переглянути повідомлення. Додаток відкритий для аудитування, що дозволяє експертам перевірити його безпеку [7].
- WhatsApp – популярний месенджер з більш ніж 2 мільярдами користувачів по всьому світу. WhatsApp пропонує не тільки текстові повідомлення, але й голосові дзвінки, відео-дзвінки, аудіо та відео повідомлення. Додаток також має шифрування повідомлень на рівні клієнта, що забезпечує конфіденційність.
- Telegram – месенджер, який також пропонує голосові повідомлення, текстові повідомлення, групові чати та відео-дзвінки. Telegram пропонує шифрування повідомлень, але відкритий код додатка не повністю перевірений на безпеку.
- Viber – це месенджер з більш ніж 1 мільярдом користувачів. Він пропонує голосові дзвінки, відео-дзвінки, аудіо та відео повідомлення, а також проводить шифрування повідомлень на рівні клієнта.

1.3.1 Telegram

Telegram - це популярний месенджер, який був створений у 2013 році Павлом Дуровим і його братом Ніколаєм. Telegram відомий своєю високою швидкістю та безпекою, а також функціями, які роблять його унікальним серед інших месенджерів. Однією з основних переваг Telegram є його криптографічна безпека. Telegram використовує протокол шифрування MTProto, який забезпечує захист від перехоплення повідомлень третіми особами. Крім того, Telegram дозволяє користувачам шифрувати свої повідомлення з використанням приватного ключа, що забезпечує ще більшу безпеку [8].

Також важливою функцією Telegram є можливість створювати секретні чати, в яких повідомлення автоматично видаляються через певний час після того, як їх було прочитано. Це дозволяє забезпечити повну конфіденційність розмов.

Ще однією важливою функцією Telegram є можливість створювати групи з до 200 000 учасників. Це робить його ідеальним для великих спільнот, таких як команди проектів, спільноти фанатів, релігійні організації та інші.

Окрім текстових повідомлень, Telegram також дозволяє користувачам відправляти голосові повідомлення, відео, фото, документи та інші файли. Також Telegram має вбудований інструмент для розсилання опитувань та голосувань, що дозволяє легко збирати відгуки від користувачів.

Шифрування в телеграм є однією з ключових функцій цього месенджера, яка забезпечує конфіденційність та безпеку передачі повідомлень. У Телеграмі використовується протокол шифрування MTProto, який є внутрішнім алгоритмом шифрування, розробленим командою телеграм. Цей протокол забезпечує end-to-end шифрування, що означає, що дані шифруються на пристрої відправника і розшифровуються тільки на пристрої отримувача [9].

Процес шифрування в телеграмі включає генерацію симетричного ключа шифрування для кожного окремого чату, який використовується для зашифровування та розшифрування повідомлень. Цей ключ шифрується асиметричним шифруванням з використанням RSA або Diffie-Hellman, залежно від версії протоколу MTProto [10].

На рисунках 1.2, 1.3 проілюстровано інтерфейси користувачів на різних платформах.

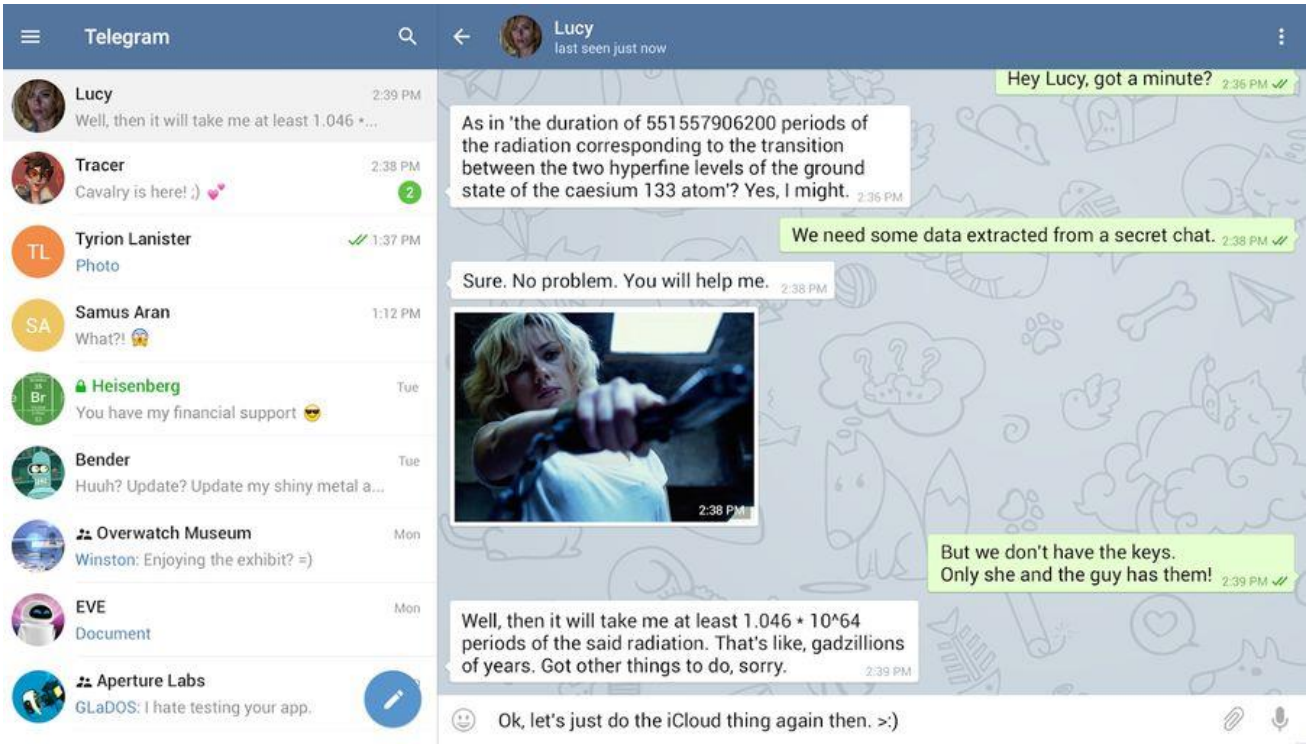

Рисунок 1.2 – Інтерфейс користувача на мобільній платформі Android

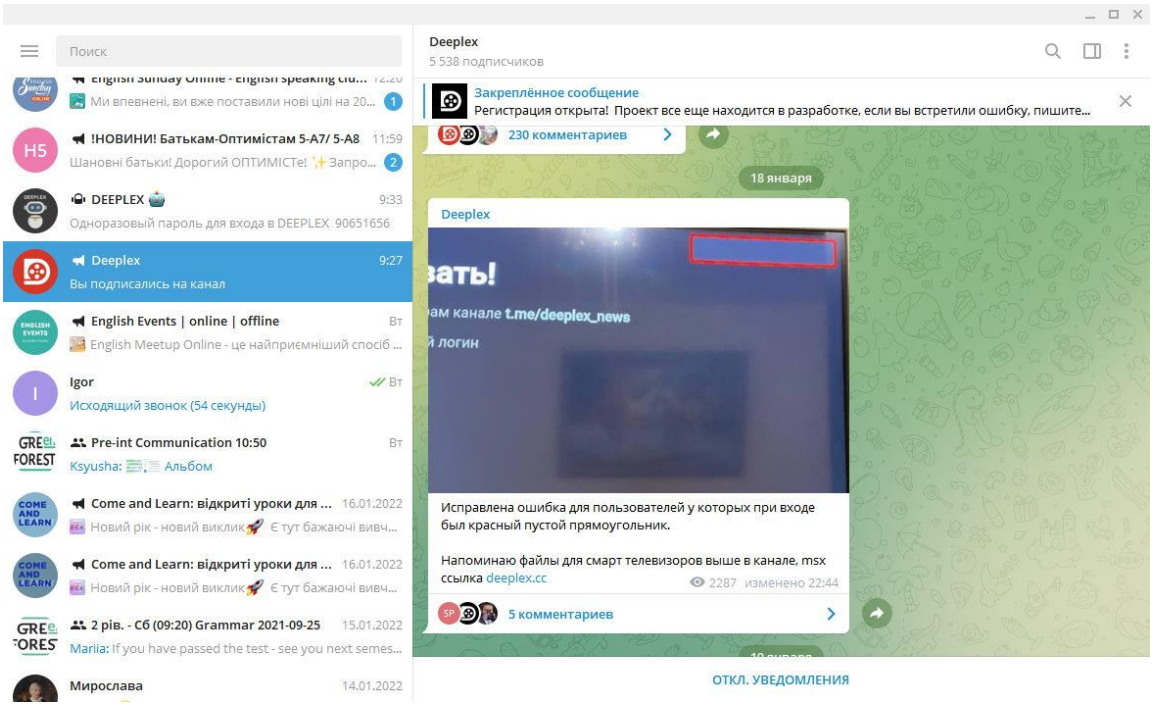

Рисунок 1.3 – Інтерфейс користувача на платформі Windows

1.3.2 WhatsApp

WhatsApp є одним з найпопулярніших месенджерів у світі, що дозволяє користувачам обмінюватись текстовими повідомленнями, голосовими повідомленнями, зображеннями, відео та документами.

WhatsApp був створений у 2009 році американським розробником Браяном Ектоном, але згодом його було придбано компанією Facebook у 2014 році за 19 мільярдів доларів. З того часу WhatsApp став частиною корпорації Facebook і на даний момент має понад 2 мільярди активних користувачів у всьому світі [11].

Однією з основних переваг WhatsApp є його безкоштовність. Користувачі можуть відправляти повідомлення один одному, не сплачуючи за це гроші. Крім того, WhatsApp пропонує шифрування повідомлень для забезпечення безпеки користувачів.

Ще однією важливою функцією WhatsApp є його можливість використовуватись не тільки на мобільних пристроях, але й на комп'ютерах. Користувачі можуть відкривати свій обліковий запис на різних пристроях, що дозволяє їм зручно комунікувати зі своїми контактами.

Однак, у порівнянні з Telegram, WhatsApp має деякі обмеження щодо розміру файлів, які можна надіслати, а також не має функцій для відправки повідомлень з автозниженням терміну дії. Також у WhatsApp були проблеми з конфіденційністю даних, що спричинило критику користувачів та регуляторних органів. Однак, компанія Facebook постійно покращує свої політики щодо конфіденційності та захисту даних користувачів.

Шифрування в WhatsApp є однією з основних функцій цього популярного месенджера, спрямованою на забезпечення конфіденційності та безпеки повідомлень. У WhatsApp, як і в Telegram використовується end-to-end шифрування [12].

Під час процесу шифрування в WhatsApp використовується протокол шифрування Signal, який є відкритим стандартом для забезпечення безпеки комунікацій. Цей протокол використовує симетричне шифрування з симетричним ключем, який генерується на пристроях користувачів [13].

На рисунках 1.4, 1.5 проілюстровано інтерфейси користувачів на різних платформах.

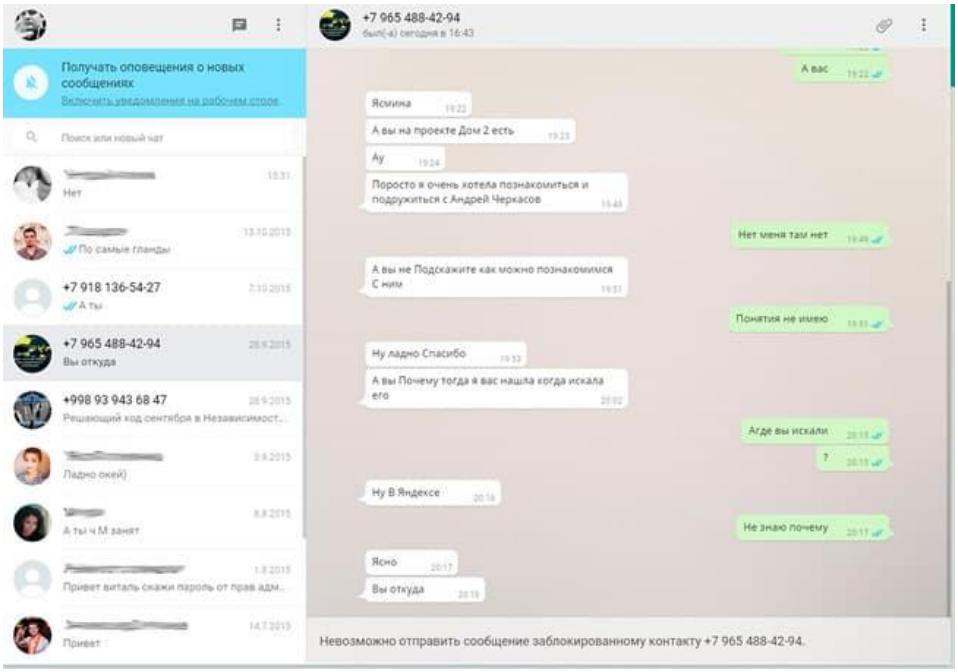

Рисунок 1.4 – Інтерфейс користувача на платформі Windows

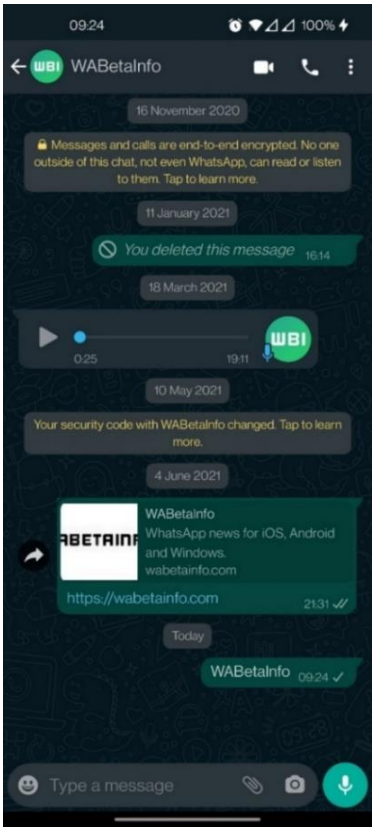

Рисунок 1.5 – Інтерфейс користувача на мобільній платформі Android

1.3.3 Viber

Viber – це ще один популярний месенджер, який був запущений в 2010 році. Він дозволяє користувачам обмінюватися текстовими, голосовими і відео повідомленнями, а також викликами через Інтернет.

Основні функції Viber:

- Текстові повідомлення: Viber дозволяє відправляти текстові повідомлення, які можуть бути звичайними або зашифрованими з використанням протоколу end-to-end encryption (E2EE), що унеможливлює доступ до даних третім особам, включаючи провайдерів послуг чи розробників месенджерів [14].
- Голосові і відео дзвінки: Viber підтримує голосові і відео дзвінки через Інтернет, що дозволяє користувачам зв'язуватися з іншими людьми у будь-якій точці світу.
- Стікери та емодзі: Viber має велику колекцію стікерів та емодзі, які можна використовувати для вираження емоцій під час обміну повідомленнями.
- Відправлення файлів: Viber дозволяє відправляти файли будь-якого типу і розміру, включаючи фото, відео, документи та інші.
- Групові чати: Viber підтримує групові чати до 250 учасників.
- Public Chats: Viber дозволяє створювати та приєднуватися до публічних чатів з відомими людьми, брендами та медіа-компаніями.
- Viber Out: це функція, яка дозволяє користувачам здійснювати дзвінки на мобільні та стаціонарні телефони за міжнародними тарифами.

Viber є безкоштовним месенджером, який можна встановити на мобільні пристрої з операційними системами Android, iOS, Windows Phone, а також на ПК. Він також має вбудовану функцію шифрування повідомлень.

В Viber шифрування використовується для забезпечення приватності та безпеки комунікації між користувачами. Вбудована функція шифрування, відома як Viber Security, захищає повідомлення та дзвінки від несанкціонованого доступу [15].

Шифрування в Viber базується на протоколі шифрування Signal, який вважається одним з найбільш безпечних протоколів шифрування у сфері месенджерів. Він використовує симетричне шифрування з випадковими симетричними ключами, що генеруються на пристроях користувачів, та асиметричне шифрування з використанням асиметричних ключів.

На рисунках 1.6, 1.7 проілюстровано інтерфейси користувачів на різних платформах.

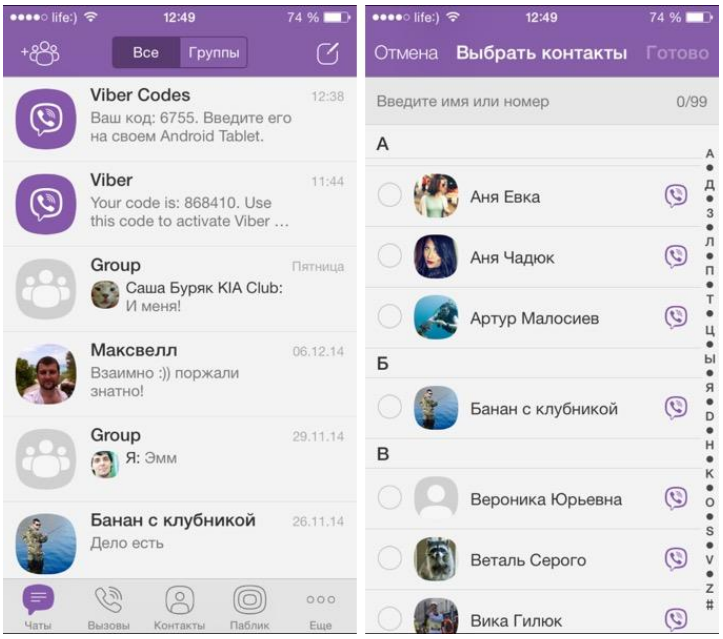

Рисунок 1.6 – Інтерфейс користувача на мобільній платформі Android

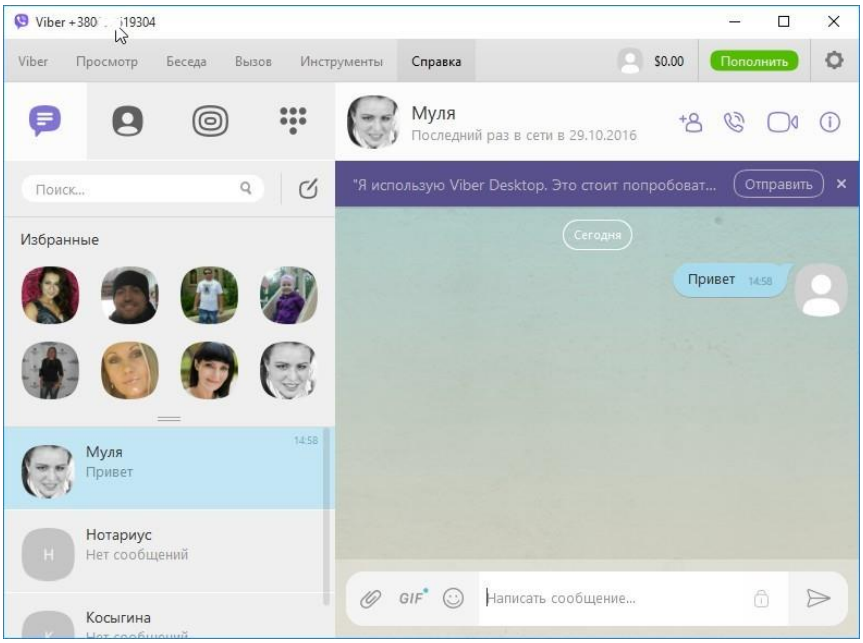

Рисунок 1.7 – Інтерфейс користувача на платформі Windows

1.3.4 Signal

Signal – це месенджер з відкритим вихідним кодом, розроблений в компанії Signal Foundation і заснований на протоколі Signal Protocol. Головною метою розробників була забезпечити високий рівень безпеки та приватності користувачів [16].

Основні особливості Signal:

1. Шифрування. Всі повідомлення в Signal захищені точкою-точкою шифруванням, який є одним з найбільш надійних методів захисту даних від несанкціонованого доступу.
2. Немає можливості збереження повідомлень на сервері. Це означає, що навіть якщо сервер буде зламаний, зловмисники не зможуть отримати доступ до повідомлень користувачів.
3. Анонімність номера телефону. Користувачі можуть зареєструватись в Signal, використовуючи лише номер телефону без необхідності надавати особисті дані.
4. Груповий чат. Signal підтримує створення групових чатів з можливістю відправляти текстові повідомлення, відео, фото та інші типи файлів.
5. Знищення повідомлень. Користувачі можуть встановлювати термін, протягом якого повідомлення будуть зберігатись на пристрої отримувача, після чого вони будуть автоматично видалені.

Signal є досить популярним месенджером, особливо серед тих, хто цінує конфіденційність та безпеку своїх даних.

Signal використовує протокол шифрування Signal Protocol, який розроблений з урахуванням сучасних криптографічних стандартів та найкращих практик безпеки. Цей протокол використовує симетричне шифрування з випадковими ключами для захисту змісту повідомлень, а також асиметричне шифрування для обміну ключами та підтвердження автентичності [17]. При обміні повідомленнями або викликами, Signal автоматично генерує сеансовий ключ, який використовується для шифрування

та розшифрування даних. Ці ключі автоматично обмінюються між користувачами безпосередньо та захищаються від зловмисників [18].

На рисунках 1.8, 1.9 проілюстровано інтерфейси користувачів на різних платформах.

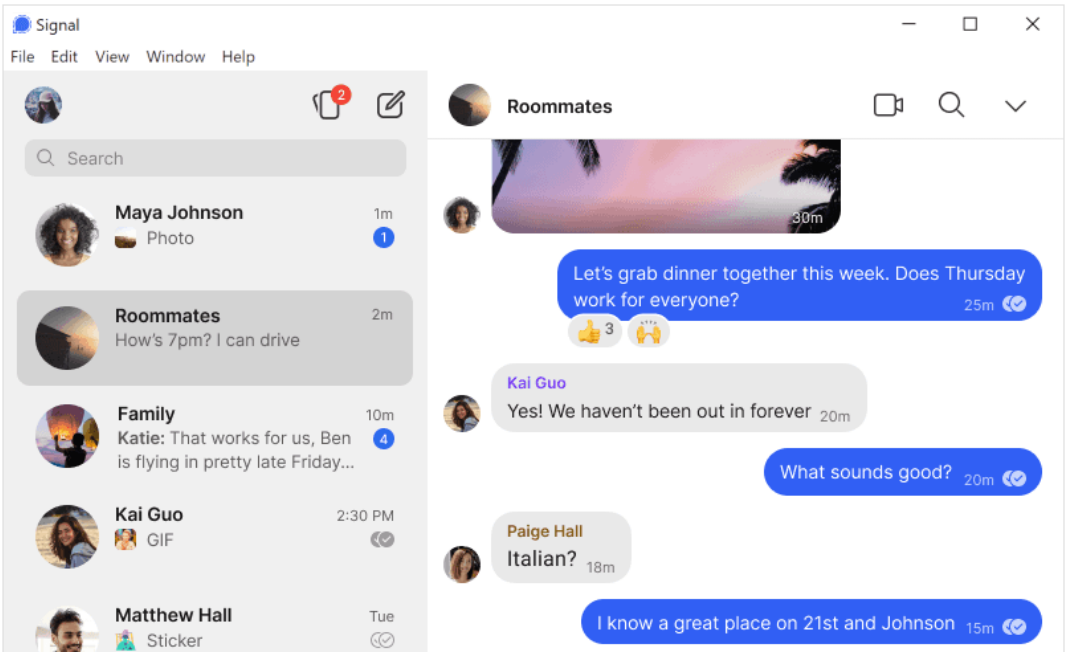

Рисунок 1.8 – Інтерфейс користувача на платформі Windows

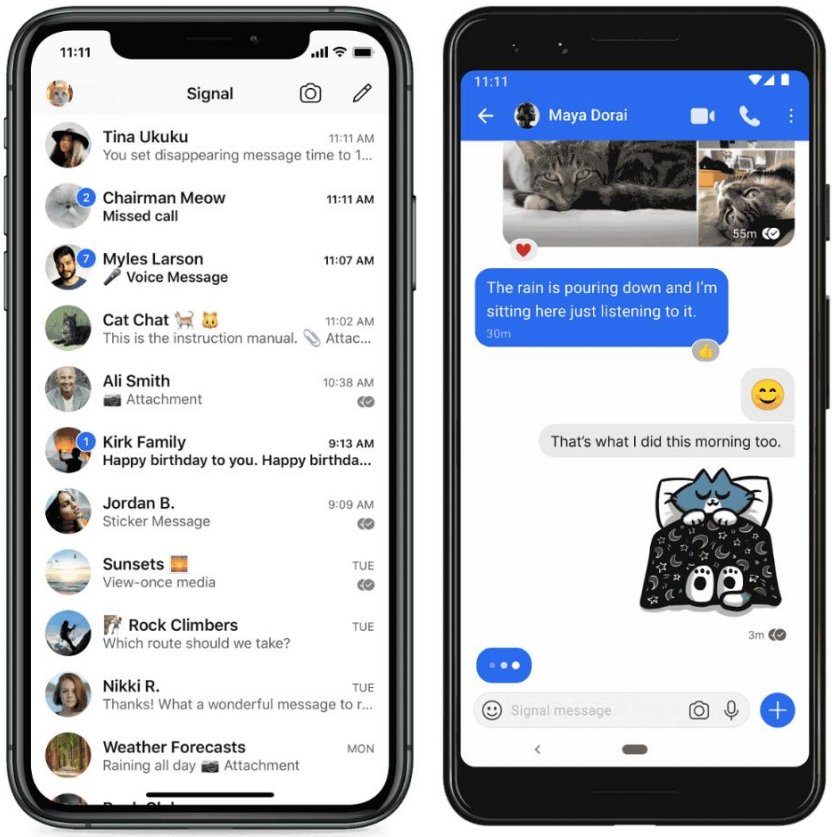

Рисунок 1.9 – Інтерфейс користувача на мобільній платформі Android

Нижче наведена таблиця 1.1, що порівнює месенджери-аналоги за їх функціональністю.

Таблиця 1.1

Порівняння існуючих аналогів

	WhatsApp	Viber	Telegram	Signal
Наскрізне шифрування чатів та дзвінків за замовчуванням	+	+	- Тільки для секретних чатів	+
Всі повідомлення видаляються після доставки та не зберігаються на серверах	+	+	+	+
Захист від скріншотів	-	+	+ Тільки для секретних чатів	-
Видалення відправленого повідомлення для всіх учасників чату	+	+	+	-
Групові аудіо- та відеодзвінки	+	+	-	+
Відправка файлів, фото, відео	+	+	+	+
Завантаження великих файлів (1 гб+)	-	-	+	-
Відкритий вихідний код	-	-	-	+
Максимальна кількість учасників у групових чатах	256	250 в закритих групах/не обмежено в спільнотах	200000	150

Конфіденційність та безпека в обміні голосовими повідомленнями є дуже важливими, особливо якщо вони містять особисту або конфіденційну інформацію. За звичайних обставин користувачі мають небагато контролю над тим, як їхні голосові повідомлення зберігаються, передаються та обробляються.

З цієї причини, важливо розробляти програмний застосунок для конфіденційного обміну голосовими повідомленнями, який забезпечуватиме безпеку користувачів та конфіденційність їхніх даних.

Такий застосунок повинен мати такі функції, як шифрування повідомлень, автентифікацію користувачів, перевірку цілісності повідомлень та можливість контролювати терміни зберігання повідомлень.

1.4 Висновки

У даному розділі було досліджено поняття месенджерів та їх класифікацію залежно від типу, принципу роботи та особливостей функціонування. Були розглянуті чотири найпопулярніших месенджера: Telegram, WhatsApp, Viber та Signal. Кожен з них має свої переваги та недоліки, але загалом є досить функціональними та зручними для користування.

Дослідження показали, що месенджери стали невід’ємною частиною життя сучасної людини та є важливим інструментом комунікації як в особистому, так і в бізнес-середовищі. Було виявлено, що найбільшої уваги серед користувачів набуває приватність та конфіденційність інформації, що обмінюється через месенджери. Більшість з досліджених месенджерів мають захист від перехоплення та шифрування повідомлень, але не всі з них гарантують абсолютну конфіденційність та захист персональних даних.

Отже, знання про особливості та функціональні можливості різних типів месенджерів є важливим для розробки програмного забезпечення, що міститиме функцію конфіденційного обміну голосовими повідомленнями.

У наступному розділі будуть описані технології та методи, які можна використовувати для розробки програмного застосунку для конфіденційного обміну голосовими повідомленнями.

РОЗДІЛ 2
ОБРАНИЙ СТЕК ТЕХНОЛОГІЙ

Для розробки месенджеру використовувались декілька мов програмування: NodeJS, Java, Kotlin. В якості IDE використана Visual Studio Code для NodeJS та Andoid Studio для Java та Kotlin. Для операцій створення та зберігання в базі даних була обрана СУБД Firebase.

В наш час технології прийнято використовувати разом у зв'язку зі зручністю. Тому Firebase була обрана у зв'язку з простотою використання при написанні коду NodeJS. А Java та Kotlin сумісні та прості у використанні. Для тестування використовувалася програма Android Studio.

Для розробки серверної частини використовувався NodeJS, розробки клієнтської частини використовувалася середовище Android Studio. Для тестування готової програми використовувалися смартфони, що були в наявності.

2.1 Архітектура побудови додатку

Архітектура типу клієнт-сервер (рисунок 2.1) є однією з найпоширеніших архітектур для месенджерів та багатьох інших додатків. В цій архітектурі розробка розподіляється між клієнтською і серверною сторонами, які взаємодіють між собою для забезпечення функціональності та обміну даними [19].

Основні компоненти архітектури клієнт-сервер включають:

1. Клієнтська сторона: це додаток, який встановлюється на пристроях користувачів (наприклад, смартфони, комп'ютери). Клієнтська сторона надає інтерфейс користувача для взаємодії з месенджером. Вона відповідає за збір та відображення даних, обробку введення користувача та відправку запитів до серверної сторони.

2. Серверна сторона: це центральний компонент, який обробляє запити від клієнтської сторони, зберігає та керує даними, забезпечує функціональність

месенджера. Серверна сторона може включати різні компоненти, такі як сервер додатків, база даних, системи безпеки та інші. Вона відповідає за обробку запитів, збереження та передачу даних між клієнтами.

3. Протоколи комунікації: для взаємодії між клієнтською та серверною сторонами використовуються протоколи комунікації, такі як HTTP, WebSocket тощо. Ці протоколи визначають правила та формат обміну даними між клієнтом та сервером.

4. Безпека: у месенджерах безпека грає важливу роль. Для забезпечення конфіденційності та безпеки обміну повідомленнями можуть застосовуватись різні методи шифрування, автентифікації та контролю доступу.

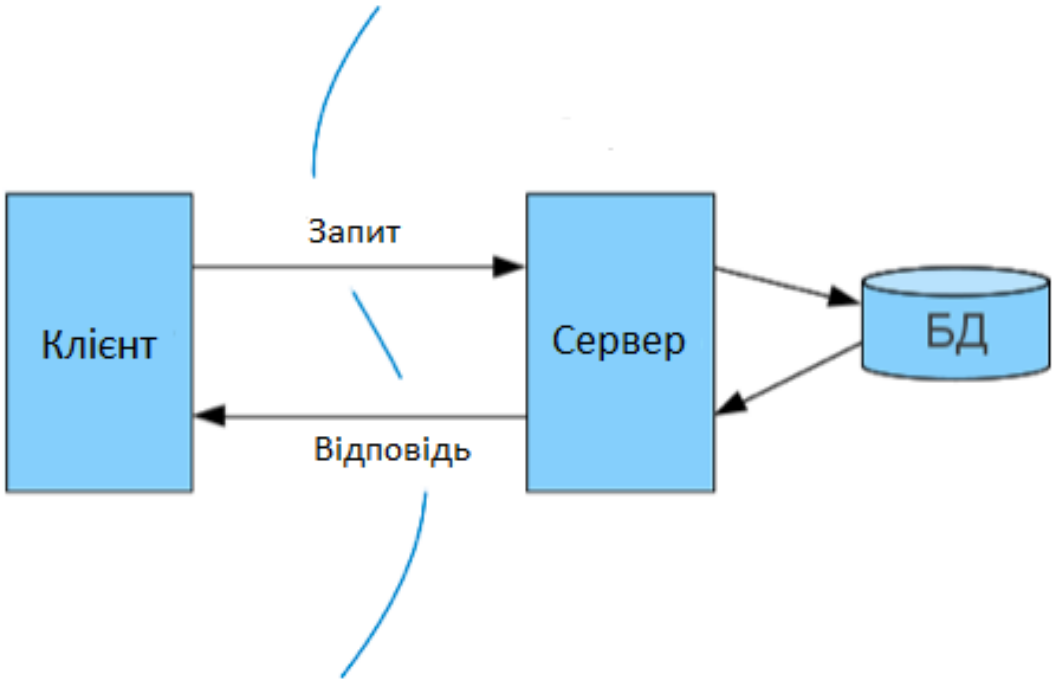

Рисунок 2.1 – Загальна схема архітектури типу «Клієнт-Сервер»

Архітектура клієнт-сервер має свої плюси та мінуси. До плюсів можна віднести:

1. Розділення завдань: клієнтська сторона та серверна сторона виконують різні функції, що дозволяє зосередитись на конкретних завданнях. Це полегшує розробку та підтримку додатку.

2. Масштабованість: завдяки розділенню між клієнтом та сервером, можна легко масштабувати систему, додавати нові сервери або розподіляти навантаження, що дозволяє обробляти більше запитів та обслуговувати більше користувачів.

3. Забезпечення безпеки: за допомогою серверної сторони можна контролювати доступ та виконувати заходи безпеки, такі як автентифікація, авторизація та шифрування даних. Це дозволяє забезпечити конфіденційність та цілісність інформації.

4. Швидкодія: клієнтська сторона може бути оптимізована для швидкого відгуку та відображення даних, тоді як серверна сторона може забезпечити потужну обробку та зберігання великих обсягів даних .

З мінусів можна виділити:

1. Залежність від сервера: у випадку відмови сервера або проблем з мережевим з'єднанням, користувачі можуть бути обмежені у функціональності або не мати доступу до даних.

2. Вимоги до пропускної здатності мережі: клієнтська сторона потребує стабільного та швидкого з'єднання з сервером для обміну даними. Погана якість мережі може призвести до затримок або втрати з'єднання.

3. Складність розгортання: розгортання серверної інфраструктури та її налагодження можуть вимагати додаткових зусиль та ресурсів.

4. Конфігурація та сумісність: клієнтська сторона та серверна сторона повинні бути налаштовані та сумісні між собою, що може створювати складнощі при оновленні або зміні технологій [20-24].

2.2 Visual Studio Code

Visual Studio Code (VS Code) – це популярний та широко використовуваний текстовий редактор, розроблений компанією Microsoft. Це легкий і універсальний інструмент, призначений для задоволення потреб розробників у різних мовах програмування та на різних платформах.

VS Code (рисунок 2.2) надає широкий набір функцій, які поліпшують процес розробки. Він підтримує виділення синтаксису, інтелектуальне завершення коду, можливості налагодження, інтеграцію з системами контролю версій та багато іншого. У ньому також є можливість використовувати розширення, які дозволяють розширити його функціональність та пристосувати його до своїх потреб.

Visual Studio Code також володіє декількома перевагами, які зробили його популярним серед розробників:

- Легкість використання: VS Code має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє розробникам швидко орієнтуватися та почати працювати без зайвих зусиль.
- Розширюваність: завдяки великому екосистемі розширень, розробники можуть розширювати можливості VS Code за допомогою різноманітних плагінів і розширень, які додають нові функції та інструменти.
- Інтеграція з системами контролю версій: VS Code має підтримку популярних систем контролю версій, таких як Git, що дозволяє легко взаємодіяти з репозиторіями та керувати версіями коду.
- Потужний редактор: VS Code надає можливості редагування коду, такі як автодоповнення, виправлення помилок, переміщення по коду та багато іншого. Це допомагає розробникам писати код швидше та ефективніше.
- Вбудовані інструменти для налагодження: VS Code має вбудовану підтримку для налагодження коду, що дозволяє розробникам встановлювати точки зупинки, крокувати по коду та аналізувати значення змінних для виявлення та виправлення помилок.

VS Code є крос-платформеним редактором і доступний для використання на ОС Windows, macOS та Linux. Це дозволяє розробникам працювати в улюбленому середовищі на будь-якій платформі. Через його простий та інтуїтивно зрозумілий інтерфейс, VS Code стає вибором багатьох розробників для щоденної роботи над проектами програмного забезпечення [25-27].

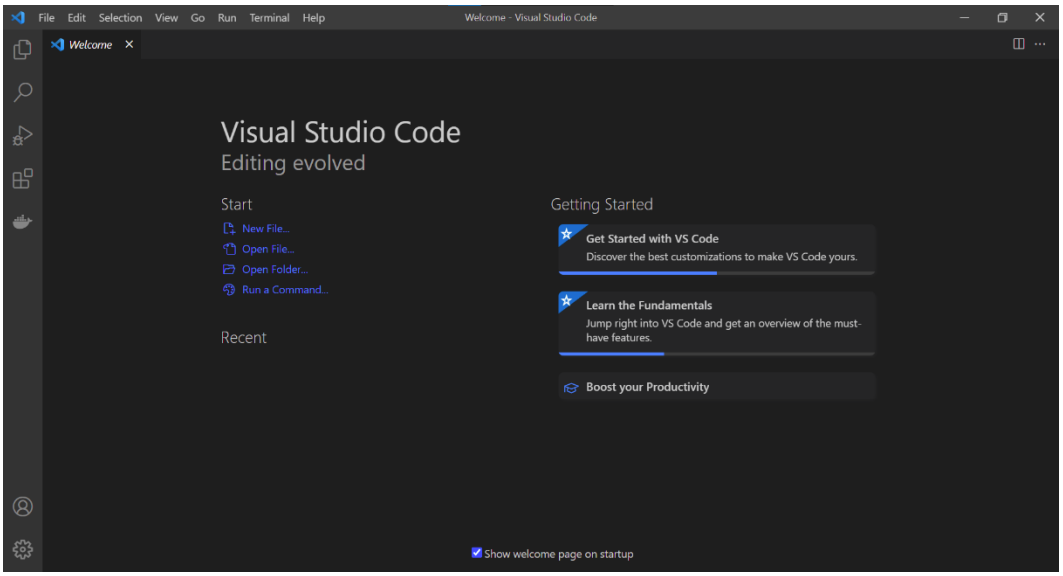

Рисунок 2.2 – Інтерфейс програми Visual Studio Code

2.3 NodeJS

Node.js є серверним середовищем виконання, яке дозволяє використовувати JavaScript для розробки серверних додатків. Однією з ключових особливостей Node.js є його асинхронна та подієво-орієнтована модель програмування. Це означає, що він може обробляти багато запитів одночасно, не блокуючи виконання інших операцій.

Середовище базується на двигуні V8, розробленому компанією Google для браузера Chrome. Він надає високу швидкодію та продуктивність, що робить його ефективним для обробки великого обсягу запитів. Node.js також має вбудовану підтримку модулів, що дозволяє легко розширювати його функціональність.

Node.js (рисунок 2.3) є платформою з великою спільнотою розробників, яка активно підтримує його розвиток та надає різноманітні модулі та пакети для спрощення розробки. Вона також надає доступ до розширеного набору інструментів та бібліотек, що дозволяють розробникам створювати різноманітні застосунки та сервіси [28].

В цілому, Node.js може бути використаний для розробки різних типів додатків, включаючи веб-сервери, мережеві додатки, API та інші. Його гнучкість та широкі можливості зробили його популярним вибором серед розробників для створення сучасних серверних додатків.

Рисунок 2.3 – Базова архітектура Node.js

2.4 Android Studio

Android Studio є інтегрованою середовищем розробки (IDE) для створення мобільних додатків для платформи Android, що надає розробникам потужні інструменти та ресурси для ефективної розробки, тестування та налагодження додатків. Базується на популярному проекті IntelliJ IDEA і має багато функціональних можливостей, що спрощують процес розробки. Це забезпечує зручне середовище для написання коду на мові програмування Kotlin та Java, які є основними мовами для розробки Android-додатків.

Android Studio має вбудовані інструменти для розробки користувацького інтерфейсу, такі як редактор макетів, дизайнер ресурсів та палітра компонентів. Це дозволяє розробникам швидко створювати і налаштовувати вигляд та поведінку своїх додатків. Окрім цього, Android Studio включає в себе інтегровану систему збірки та пакування додатків, яка дозволяє легко створювати APK-файли для розповсюдження на пристрої Android. Середовище також надає розширені можливості для тестування додатків, включаючи інструменти для автоматичного тестування та налагодження.

Android Studio інтегрована з Android SDK, що надає доступ до різноманітних API та функціональності платформи Android. Це дозволяє розробникам

використовувати потужні можливості, такі як робота з сенсорами, камерою, мережею та іншими пристроями [29-31].

2.5 Java

Java - це потужна, універсальна мова програмування, яка використовується для розробки різноманітного програмного забезпечення. Вона була створена з метою бути "писаною один раз, запущено скрізь" (Write Once, Run Anywhere), що означає, що програми, написані на Java, можуть працювати на різних платформах, таких як комп'ютери, мобільні пристрої та вбудовані системи.

Java має декілька ключових особливостей, які роблять її популярною серед розробників. Вона має простий і зрозумілий синтаксис, що сприяє легкому вивченню та розумінню мови. Java є об'єктно-орієнтованою мовою програмування, що дозволяє розробникам створювати модульний та перевикористовуваний код.

Однією з найсильніших сторінок Java є її платформа Java Virtual Machine (JVM). JVM є віртуальною машиною, яка виконує Java-код і перетворює його на машинний код, зрозумілий операційної системі. Це дозволяє програмам, написаним на Java, працювати на будь-якій платформі, яка підтримує JVM, безпосередньо взаємодіючи з операційною системою [32].

Також ця мова має велику стандартну бібліотеку, яка включає в себе різноманітні інструменти та класи для розвитку програмного забезпечення. Ця бібліотека дозволяє розробникам швидко та ефективно вирішувати багато типових задач, таких як робота з мережевими підключеннями, робота з базами даних, обробка рядків та багато іншого.

2.6 Kotlin

Kotlin – це сучасна, статично типізована мова програмування, яка працює на віртуальній машині Java (JVM) і є альтернативою мові Java. Вона була розроблена компанією JetBrains з метою поліпшення продуктивності розробників та спрощення

процесу розробки програмного забезпечення. Kotlin поєднує в собі простий та зрозумілий синтаксис з багатьма функціональними можливостями, що робить її привабливою для розробників. Вона підтримує об'єктно-орієнтоване та функціональне програмування, що дозволяє розробникам використовувати різноманітні підходи при створенні програм.

Однією з головних переваг Kotlin є його сумісність з Java. Всі існуючі бібліотеки, фреймворки та інструменти, розроблені для Java, можуть бути використані в проектах Kotlin без будь-яких проблем. Це робить Kotlin привабливим вибором для компаній, які вже використовують Java технології і хочуть поступово перейти на більш сучасну мову програмування.

Kotlin має багатий набір функцій і властивостей, які сприяють поліпшенню продуктивності розробників. Вона підтримує нульову безпеку, що дозволяє уникнути багатьох типових помилок, пов'язаних з нульовими посиланнями. Крім того, Kotlin надає можливість використовувати короткі, збільшені методи та оператори, що полегшує читання та написання коду.

Мова також підтримує розробку мобільних додатків для платформи Android. Вона є офіційною мовою розробки для Android і надає розробникам багато переваг, таких як покращена безпека, спрощений синтаксис та підтримка функціонального програмування [33-35].

2.7 Firebase

Firebase – це платформа розробки програмного забезпечення, яка надає широкі можливості для створення високопродуктивних та масштабованих додатків та пропонує різноманітні сервіси та інструменти, які дозволяють розробникам швидко будувати та впроваджувати додатки для мобільних пристроїв, веб-сайтів та серверних середовищ.

Firebase забезпечує повну інфраструктуру для розробки додатків, включаючи рішення для аутентифікації користувачів, зберігання та синхронізації даних в реальному часі, хостингу веб-сайтів, аналітики додатків, розсилки повідомлень та

багато іншого. Завдяки інтеграції з іншими сервісами Google, Firebase надає доступ до потужних інструментів машинного навчання, аналізу даних та хмарних обчислень.

Перевагою Firebase є його простота використання та невимогливість до інфраструктури. Розробники можуть швидко налаштувати та почати використовувати сервіси Firebase без необхідності налаштовувати та керувати власним сервером. Багато рутинних задач, пов'язаних з розробкою додатків, вже вирішені в Firebase, що дозволяє зосередитись на розробці функціоналу [36].

Firebase підтримує різні платформи, включаючи Android, iOS, веб та серверні середовища. Інтеграція Firebase з різними мовами програмування та фреймворками дуже проста, що робить його універсальним рішенням для розробки додатків на різних платформах. На рисунку 2.4 зображено можливості Firebase, що будуть використані в процесі розробки застосунку.

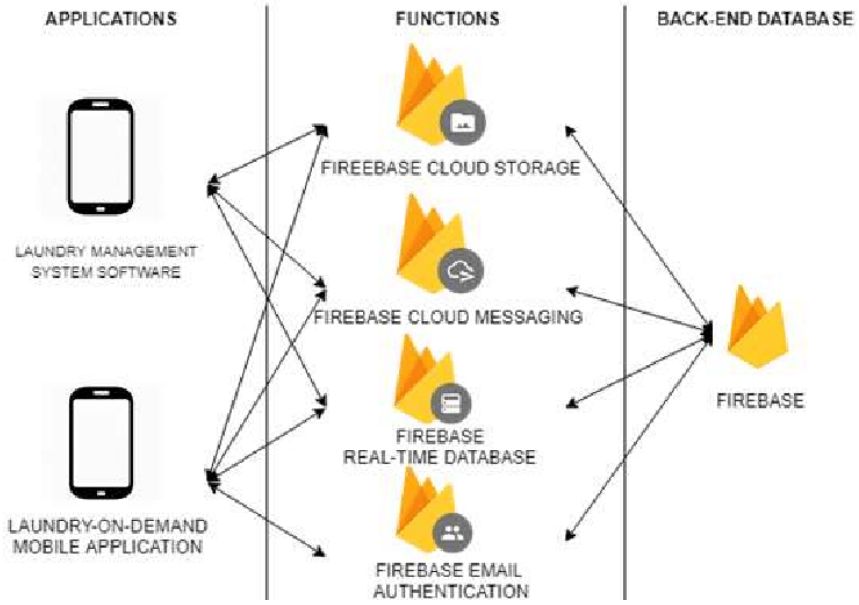

Рисунок 2.4 – Можливості Firebase

2.8 Висновки

У даному розділі було проведено аналіз технологій, що використовуються для розробки месенджерів. Були розглянуті такі технології як NodeJS, Java, Kotlin, Visual Studio Code, Android Studio та Firebase.

NodeJS є популярним серед розробників серверних додатків, оскільки він базується на JavaScript і дозволяє швидко створювати ефективні мережеві додатки.

Java та Kotlin – мови програмування, що широко використовуються для розробки мобільних додатків на платформі Android. Вони надають потужні можливості та добре підтримуються Android-екосистемою.

Visual Studio Code та Android Studio – популярні інтегровані середовища розробки (IDE), які надають розширені функціональні можливості та допомагають підвищити продуктивність розробки.

Firebase є хмарною платформою, яка надає широкий спектр сервісів, необхідних для розробки месенджерів, включаючи зберігання даних, аутентифікацію користувачів та реалізацію реального часу.

Використання цих технологій дозволяє розробникам побудувати потужні та ефективні месенджери з різноманітними функціями та можливостями.

Для успішного розроблення месенджера необхідно враховувати особливості кожної технології, їхню сумісність та використовувати їх належним чином для досягнення поставлених цілей проекту.

Після проведеного аналізу технологій можна перейти до третього розділу, в якому будуть розглянуті основні етапи розробки месенджера та реалізація конфіденційного обміну голосовими повідомленнями.

РОЗДІЛ 3
РОЗРОБКА МЕСЕНДЖЕРУ

У цьому розділі буде розглянуто реалізацію месенджеру з використанням обраного стеку технологій, таких як NodeJS, Java, Kotlin і Firebase.

Архітектура системи базується на клієнт-серверній моделі. Клієнтська частина відповідає за взаємодію з користувачем, а серверна частина забезпечує обробку запитів, збереження даних та передачу повідомлень між користувачами. Використовуються мови програмування NodeJS, Java і Kotlin для розробки компонентів системи.

Firebase використовується як база даних та інструмент для збереження даних користувачів. Вона надає широкі можливості для маніпуляції з даними, включаючи збереження повідомлень, керування користувачами та забезпечення конфіденційності даних.

В рамках реалізації месенджеру розробляються функціональність обміну повідомленнями, створення профілю користувача, можливість додавання контактів, реєстрація та аутентифікація користувачів. Шифрування голосових повідомлень забезпечується за допомогою алгоритму AES.

У наступних підрозділах будуть детальніше розглянуті технічні аспекти реалізації месенджеру, включаючи опис роботи з базою даних Firebase, розробку клієнтської та серверної частин програми, а також інтеграцію шифрування голосових повідомлень.

3.1 Архітектура системи

В даному підрозділі буде розглянута архітектура системи месенджера, яка базується на клієнт-серверній моделі. Клієнтська частина відповідає за взаємодію з користувачем і забезпечує графічний інтерфейс, обробку користувацьких запитів та

відображення даних. Серверна частина відповідає за обробку запитів від клієнтів, збереження та управління даними, а також передачу повідомлень між користувачами.

Архітектура системи передбачає використання трьох мов програмування: NodeJS, Java та Kotlin. NodeJS використовується для реалізації серверної частини, яка працює на базі Node.js з використанням фреймворка Express.js. Java та Kotlin використовуються для розробки клієнтських додатків для платформ Android та JavaFX відповідно.

Основна логіка системи розподілена між клієнтською та серверною частинами (рисунок 3.1). Клієнтська частина відповідає за реалізацію функціональності, такої як створення профілю користувача, обмін повідомленнями, додавання контактів тощо. Серверна частина забезпечує обробку запитів від клієнтів, збереження та витягування даних з бази даних Firebase, а також механізми аутентифікації та авторизації користувачів.

Зв'язок між клієнтською та серверною частинами відбувається за допомогою HTTP-запитів. Клієнтська сторона відправляє запити на сервер, а сервер обробляє ці запити та повертає необхідні дані у відповідь. Для передачі реального часу повідомлень використовується також механізм веб-сокетів, який забезпечує бідиректорний зв'язок між клієнтами та сервером.

Firebase SDK дозволяє легко і швидко інтегрувати функціональність Firebase в свої додатки, забезпечуючи доступ до різних сервісів Firebase, таких як аутентифікація користувачів, база даних в реальному часі, зберігання файлів, хмарні функції та багато іншого. Firebase Admin SDK надає розширені можливості для взаємодії з Firebase з серверної сторони. Цей набір інструментів дозволяє створювати потужні серверні додатки, які використовують функціонал Firebase для роботи з даними, аутентифікації та сповіщень [37-39].

Архітектура системи месенджера розроблена з метою забезпечення швидкості, масштабованості та надійності. Компоненти системи розподілені таким чином, що дозволяють збільшувати потужність серверів та кількість клієнтів залежно від потреб проекту. Крім того, використання Firebase як бази даних дозволяє забезпечити стійкість даних та їхню реплікацію для забезпечення доступності.

Рисунок 3.1 – Базова схема роботи застосунку

3.2 Компоненти системи

У цьому підрозділі будуть розглянуті основні компоненти системи месенджера і їхні функціональні можливості.

1. Реєстрація користувачів: даний компонент відповідає за реєстрацію нових користувачів у системі. Він надає можливість створити обліковий запис, вказавши необхідну інформацію, таку як ім'я, електронна пошта та пароль. Після реєстрації, дані про користувача зберігаються в базі даних Firebase.

2. Аутентифікація користувачів: даний компонент відповідає за аутентифікацію користувачів при вході в систему. Він перевіряє правильність введеного пароля та ідентифікатора користувача і надає доступ до особистого облікового запису. Для забезпечення безпеки, дані для аутентифікації передаються по захищеному з'єднанню.

3. Список контактів: даний компонент дозволяє користувачам управляти своїми контактами. Вони можуть додавати нові контакти, видаляти існуючі та переглядати інформацію про них. Список контактів синхронізується між всіма пристроями користувача і зберігається в базі даних Firebase.

4. Обмін повідомленнями: цей компонент дозволяє користувачам обмінюватись текстовими повідомленнями. Користувачі можуть створювати нові розмови, відправляти повідомлення та отримувати їх від інших учасників. Повідомлення передаються через сервер за допомогою веб-сокетів, що забезпечує швидку доставку і реальний час обміну повідомленнями.

5. Голосові повідомлення: цей компонент дозволяє користувачам обмінюватись голосовими повідомленнями. Вони можуть записувати свої голосові повідомлення, відправляти їх і прослуховувати отримані повідомлення. Голосові повідомлення зберігаються в зашифрованому вигляді для забезпечення конфіденційності.

6. Налаштування профілю: даний компонент дозволяє користувачам налаштовувати свій особистий профіль. Вони можуть змінювати своє ім'я, фото профілю та інші персональні дані. Зміни в профілі синхронізуються між всіма пристроями користувача і зберігаються в базі даних Firebase [40-43].

3.3 Технічна реалізація компонентів

У цьому підрозділі буде розглянуто технічну реалізацію основних компонентів системи месенджера. Кожен компонент має свою функціональність і вимагає певних технологій та інструментів для своєї реалізації. Для забезпечення ефективної комунікації та обміну даними між компонентами будуть використовуватись відповідні протоколи та архітектурні підходи.

1. Реєстрація користувачів: для реалізації функціональності реєстрації користувачів буде використано серверну частину, написану на Node.js. За допомогою фреймворка Express будуть налаштовані маршрути та обробники HTTP-запитів для

реєстрації нових користувачів. Дані про користувачів будуть зберігатись у базі даних Firebase.

2. Аутентифікація користувачів: для аутентифікації користувачів буде використано Firebase Authentication, який надає зручний спосіб перевірки правильності введеного пароля та ідентифікатора користувача. Після успішної аутентифікації, користувач матиме доступ до свого особистого облікового запису.

3. Список контактів: список контактів користувачів буде зберігатись у базі даних Firebase. Для зручного управління контактами буде реалізовано відповідний API на серверній стороні з використанням Node.js та Express. Клієнтська сторона (Android додаток) буде використовувати відповідні HTTP-запити для отримання, додавання та видалення контактів.

4. Обмін повідомленнями: для обміну текстовими повідомленнями буде використано WebSocket протокол, що забезпечує двосторонню комунікацію між клієнтом та сервером. Серверна частина на Node.js використовуватиме бібліотеку Socket.IO для реалізації WebSocket сервера. Клієнтська сторона, яка реалізована на платформі Android з використанням Kotlin, також використовуватиме Socket.IO для здійснення з'єднання та обміну повідомленнями з сервером.

5. Голосові повідомлення: для реалізації голосових повідомлень буде використана бібліотека Bouncy Castle для шифрування голосових даних з використанням алгоритму AES. Клієнтська сторона здійснюватиме запис голосу, який буде шифруватись та відправлятись на сервер за допомогою WebSocket з'єднання. На серверній стороні голосове повідомлення розшифровуватиметься та зберігатиметься в базі даних Firebase.

6. Синхронізація повідомлень: для забезпечення синхронізації повідомлень між різними пристроями користувача буде використана Firebase Realtime Database. Коли користувач відправляє або отримує нове повідомлення, дані будуть автоматично синхронізуватись між всіма підключеними пристроями через Firebase. Це забезпечить безперервну та однакову інформацію про повідомлення на всіх пристроях користувача.

7. Оповіщення: для надсилання оповіщень про нові повідомлення або події в месенджері буде використана Firebase Cloud Messaging (FCM). При наявності нового повідомлення або іншої важливої події, серверна частина буде надсилати повідомлення до FCM, яке потім буде доставлено до відповідного пристрою користувача. Клієнтська сторона отримає оповіщення і показуватиме сповіщення користувачу.

8. Хмарне зберігання файлів: для зберігання та обміну медіа-файлами, такими як фотографії та відео, буде використана Firebase Cloud Storage. Клієнтська сторона зможе завантажувати файли на хмарне сховище, а інші користувачі зможуть переглядати та завантажувати ці файли. Firebase Cloud Storage забезпечує швидкий та безпечний доступ до файлів з будь-якого пристрою.

9. Безпека: для забезпечення безпеки комунікації та зберігання даних використовуватимуться різні методи, такі як шифрування даних, аутентифікація користувачів та забезпечення прав доступу. Firebase Authentication забезпечує безпеку авторизації та аутентифікації користувачів, а Firebase Security Rules дозволяють налаштовувати права доступу до даних та функціоналу месенджера.

10. Масштабованість: архітектура месенджера, побудована з використанням Node.js, Java та Kotlin, спроектована для масштабованості. Завдяки використанню асинхронного програмування та розподіленій системі зберігання даних Firebase, месенджер може легко розширюватись та обробляти великі обсяги повідомлень та користувачів [44-48].

3.4 Хмарна платформа

Firebase – це платформа для розробки веб- та мобільних додатків, яка надає широкий спектр інструментів та сервісів для розвитку і розгортання додатків. Вона створена компанією Google і надає розробникам зручні інструменти для роботи зі збереженням даних, аутентифікацією, хостингом, аналітикою, сповіщеннями та багатьма іншими функціями.

Нижче надані основні складові Firebase, що використовуються при розробці месенджеру:

1. Firebase Realtime Database: це база даних реального часу, яка дозволяє зберігати та синхронізувати дані між клієнтськими додатками. Вона забезпечує миттєве оновлення даних на всіх підключених пристроях.

2. Firebase Authentication: цей сервіс надає засоби для аутентифікації користувачів у додатках. Він підтримує різні методи аутентифікації, включаючи електронну пошту, соціальні мережі та інші.

3. Firebase Cloud Firestore: це гнучка й масштабована база даних документів, яка дозволяє зберігати структуровані дані для веб- та мобільних додатків. Вона підтримує розширені можливості запитів та дозволяє робити реалізацію застосунків легшою і швидшою.

4. Firebase Cloud Messaging: цей сервіс дозволяє надсилати сповіщення на мобільні пристрої з використанням хмарної інфраструктури Firebase. Він дозволяє відправляти повідомлення одному користувачеві або групі користувачів.

5. Firebase Hosting: це сервіс для хостингу веб-сторінок і додатків на серверах Firebase. Він надає зручний спосіб розгортання та оновлення веб-додатків.

6. Firebase Analytics: цей сервіс забезпечує збір та аналіз даних про використання додатків. Він дозволяє відстежувати кількість користувачів, конверсії, події та іншу аналітику, що допомагає розробникам приймати обґрунтовані рішення [49].

3.5 Серверна частина

Серверна частина месенджера є невід'ємною складовою для забезпечення його функціональності та взаємодії з клієнтською частиною. Вона відповідає за обробку запитів користувачів, збереження та управління даними, а також за забезпечення безпеки та надійності месенджеру.

Серверна частина месенджера здатна обробляти різні типи запитів, такі як реєстрація нового користувача, авторизація, відправлення та отримання повідомлень,

керування контактами, а також інші операції, пов'язані з функціональністю месенджера. Вона взаємодіє з базою даних, де зберігаються користувачі, повідомлення та інша інформація.

Серверний додаток месенджера може включати модулі для аутентифікації користувачів, шифрування даних, керування повідомленнями та інші функції. Вона також може бути відповідальною за надійну доставку повідомлень, управління сповіщеннями, синхронізацію даних між користувачами та інші аспекти, що забезпечують стабільну та ефективну роботу месенджера [50].

Під час розробки серверної частини месенджера важливо враховувати масштабованість, надійність та безпеку системи. Оптимізація запитів, використання кешування, моніторинг та логування є також важливими аспектами реалізації серверної частини месенджера.

В загальному розумінні, серверна частина месенджера відповідає за роботу всієї системи, включаючи обробку запитів, збереження та управління даними, безпеку та надійність. Вона грає важливу роль у забезпеченні функціональності месенджера та задоволенні потреб користувачів у зручному та безпечному спілкуванні.

3.6 Реалізація серверної частини

Для реалізації серверної частини месенджеру на Node.js необхідно виконати кілька кроків. Спочатку слід встановити Node.js та його залежності. Потім створити новий проект та налаштувати його.

Після створення проекту можна перейти до реалізації функцій серверної частини. Найпоширенішим способом є створення веб-сервера з використанням фреймворку, наприклад Express.js. Фреймворк допоможе зручно обробляти HTTP-запити, налаштовувати маршрутизацію та виконувати різноманітні завдання.

Для забезпечення комунікації з клієнтською частиною можна використовувати протоколи, такі як WebSocket або HTTP. WebSocket надає постійне з'єднання між клієнтом та сервером, що дозволяє в реальному часі передавати повідомлення. HTTP зазвичай використовується для передачі статичного контенту та виконання запитів.

Крім цього, слід забезпечити взаємодію з базою даних. Для цього можна використати NoSQL-базу даних, таку як Firebase Realtime Database або MongoDB. Вона забезпечує зручний спосіб зберігання та отримання даних.

Також варто приділити увагу безпеці серверної частини. Важливо захистити сервер від несанкціонованого доступу та атак. Для цього можна використовувати різні методи, такі як аутентифікація, авторизація, захист маршрутів тощо.

Після реалізації основної логіки серверної частини можна приступити до тестування та налагодження. Тестування допоможе виявити та виправити помилки. Також можна розглянути можливість розгортання сервера на хостинг-платформі для доступу до нього з будь-якого місця.

Реалізація серверної частини на Node.js для месенджеру з використанням Firebase передбачає підготовку середовища розробки та встановлення необхідних залежностей. Вона включає налаштування Firebase проекту та інтеграцію з сервером. Розробник визначає логіку обробки запитів від клієнтів та зберігання даних в Firebase базі даних. Серверна частина взаємодіє з клієнтами, забезпечуючи безпеку та автентифікацію. Це може включати перевірку ідентифікаційних даних користувачів, шифрування передачі повідомлень та забезпечення захищеного з'єднання між клієнтом і сервером. Крім того, серверна частина може включати додаткові функціональності, наприклад, реал-тайм сповіщення або перевірку наявності користувачів.. Загальна мета полягає у створенні надійного та ефективного сервера, який забезпечує плавну комунікацію між клієнтами та зберігає дані у Firebase базі даних. Для досягнення цієї мети необхідно ретельно проектувати архітектуру сервера, використовувати оптимальні алгоритми обробки запитів та ефективно взаємодіяти з Firebase для забезпечення надійності та швидкості обробки даних.

Код на мові програмування Node.js для розробки серверної частини месенджера зі зв'язком до Firebase та клієнтською частиною можна реалізувати за допомогою офіційного SDK Firebase для Node.js. Такий код представлено в Додатку А. Цей код ініціалізує з'єднання з Firebase, створює посилання на базу даних і містить функції для збереження та отримання даних.

3.7 Клієнтська частина

Клієнтська частина месенджера є важливою складовою для взаємодії користувача з месенджером. Вона відповідає за створення інтерфейсу, відображення повідомлень, керування контактами та багато іншого.

Клієнтський додаток дозволяє користувачу зручно та ефективно спілкуватися з іншими користувачами. Вона надає можливість надсилати та отримувати повідомлення, переглядати історію чатів, стежити за статусами контактів та відображати сповіщення про нові повідомлення [51].

За допомогою клієнтської частини месенджера користувач може створювати нові чати, додавати та видаляти контакти, редагувати свій профіль та встановлювати налаштування приватності. Вона також може підтримувати функції шифрування даних для забезпечення безпеки і конфіденційності повідомлень.

Клієнтська частина месенджера може бути розроблена з використанням мови програмування Kotlin або інших мов, і вона взаємодіє з серверною частиною месенджера для обміну даними та забезпечення синхронізації повідомлень та інших даних між користувачами.

3.8 Реалізація клієнтської частини

Реалізація клієнтської частини месенджеру на мові Kotlin включає створення різних екранів та компонентів, що забезпечують взаємодію з користувачем. Програміст звертає увагу на структуру програми, використовуючи паттерни проектування та принципи чистої архітектури, для забезпечення модульності та розширюваності.

Основний функціонал клієнтської частини включає авторизацію користувача, створення та управління профілем, перегляд контактного списку, обмін повідомленнями та медіафайлами, роботу зі сповіщеннями та багато іншого. Для забезпечення комунікації з сервером, програміст використовує HTTP-запити або WebSocket-з'єднання, щоб взаємодіяти з відповідними API серверної частини.

Одним з ключових аспектів реалізації є інтеграція з Firebase. Firebase надає зручні інструменти для аутентифікації користувачів, зберігання даних у реальному часі, обмін повідомленнями та управління сповіщеннями. Програміст використовує Firebase SDK для Kotlin, щоб здійснювати необхідні операції, такі як аутентифікація, зберігання та отримання даних, реєстрація слухачів змін тощо.

Для покращення користувацького досвіду, програміст використовує різноманітні бібліотеки та інструменти, доступні для Kotlin, які дозволяють створювати привабливі та інтерактивні ефекти, оптимізувати продуктивність та забезпечувати швидкий розвиток програми.

Загалом, реалізація клієнтської частини на Kotlin вимагає ретельного проектування, розуміння принципів мобільної розробки та використання відповідних інструментів та технологій для створення потужного та функціонального месенджеру.

3.9 Основні складові

Клієнтська частина месенджера складається з різних компонентів і функціональних модулів, які дозволяють користувачеві взаємодіяти з месенджером і виконувати різні операції. Основні складові клієнтської частини месенджера включають:

1. Графічний інтерфейс (UI): він забезпечує візуальне представлення месенджера користувачу. Це може бути екран авторизації, списки контактів, діалогові вікна повідомлень, налаштування та інші елементи, які дозволяють користувачеві взаємодіяти з месенджером.

2. Клієнтська логіка: це код, який відповідає за обробку подій і взаємодію з сервером. Вона включає функції для надсилання та отримання повідомлень, керування контактами, управління налаштуваннями та інші операції, які залежать від функціональності месенджера.

3. Клієнтська сторона мережевого з'єднання: ця складова відповідає за забезпечення комунікації між клієнтом і сервером. Вона включає протоколи зв'язку,

такі як HTTP або WebSocket, а також методи для відправки та отримання даних через мережу.

4. Безпека і аутентифікація: клієнтська частина також має вбудовану безпеку і механізми аутентифікації, що забезпечують захист конфіденційності та безпеки користувачів. Це можуть бути методи шифрування даних, перевірка автентичності користувача, управління дозволами і інші засоби безпеки.

5. Локальне сховище даних: клієнтська частина може зберігати деякі дані локально на пристрої користувача, наприклад, список контактів або історію повідомлень. Це дозволяє швидше завантаження та доступ до даних без необхідності постійного звернення до сервера.

3.9.1 Графічний інтерфейс

Графічний інтерфейс (UI) клієнтської частини месенджера є важливою складовою, яка взаємодіє з користувачем та забезпечує зручний спосіб навігації, взаємодії та відображення інформації. Він включає різні елементи та компоненти, які допомагають користувачеві взаємодіяти з месенджером. Графічний інтерфейс зазвичай включає в себе використання зображень, іконок, кольорів та інших візуальних елементів для створення привабливого та зручного користувальницького досвіду. Розробники використовують інструменти та фреймворки, такі як Kotlin для розробки графічного інтерфейсу клієнтської частини месенджера, що дозволяють їм швидко створювати та налаштовувати елементи інтерфейсу.

До елементів графічного інтерфейсу клієнтської частини месенджера відносять:

1. Екран авторизації: це початковий екран, на якому користувачі вводять свої облікові дані для входу в систему, наприклад, електронну пошту або номер телефону та пароль.

2. Екран реєстрації: якщо користувач ще не має облікового запису, він може зареєструватися на месенджері. Екран реєстрації зазвичай містить поля для введення необхідних персональних даних та створення облікового запису.

3. Головний екран: це основний екран месенджера, на якому користувачі бачать список контактів або розмов, а також доступ до інших функцій, таких як пошук контактів, створення нової розмови тощо.

4. Екран розмови: коли користувач вибирає конкретну розмову або повідомлення, він переходить на екран розмови. Тут відображаються повідомлення, діалоги та інші взаємодіючі елементи, такі як кнопки відправки повідомлення, прикріплення файлів тощо.

5. Додаткові екрани: крім основних екранів, месенджер може мати додаткові екрани для налаштувань, профілю користувача, додавання контактів, перегляду повідомлень тощо [52-54].

У додатку Б представлений код для екрану розмови клієнтської частини месенджера.

3.9.2 Клієнтська логіка

Клієнтська логіка в месенджері відповідає за обробку дій користувача та взаємодію з сервером. Вона включає різноманітні функції, що забезпечують коректну роботу месенджера та зручне користування ним.

Ця логіка включає в себе процеси аутентифікації, які дозволяють користувачеві увійти до системи та отримати доступ до своїх особистих повідомлень і контактів. Вона також відповідає за відображення списку повідомлень, оновлення його при отриманні нових повідомлень і сортування їх для зручного використання.

Клієнтська логіка включає функції надсилання повідомлень, які дозволяють користувачеві відправляти повідомлення до інших учасників чату. Вона обробляє введений користувачем текст повідомлення, створює відповідний об'єкт повідомлення і передає його на сервер для подальшої обробки і доставки до отримувача.

Окрім цього, клієнтська логіка здійснює оновлення стану даних, які відображаються у месенджері. Вона оновлює інформацію про статус користувачів,

оновлює інтерфейс при отриманні нових повідомлень і реагує на зміни, що відбуваються на сервері [55].

Додатково, клієнтська логіка включає управління сповіщеннями, які користувач отримує від сервера. Вона може відображати сповіщення про нові повідомлення, запити на додавання до друзів або інші важливі повідомлення, щоб користувач завжди був в курсі подій.

3.9.3 Клієнтська сторона мережевого з’єднання

Клієнтська сторона мережевого з'єднання в месенджері відповідає за встановлення та збереження зв'язку з сервером. Вона забезпечує передачу даних між клієнтом та сервером за допомогою мережевих протоколів.

Ця сторона включає функції підключення до сервера, які дозволяють клієнту встановити з'єднання з сервером месенджера. Вона використовує мережеві протоколи, такі як TCP/IP або HTTP, для ініціалізації з'єднання та встановлення надійного каналу зв'язку.

Клієнтська сторона мережевого з'єднання також включає функції передачі даних. Вона забезпечує передачу повідомлень та інших даних між клієнтом та сервером. Це може включати відправку запитів на сервер для отримання нових повідомлень, відправку введених користувачем повідомлень до сервера або отримання оновлень стану чату та користувачів в режимі реального часу.

Клієнтська сторона мережевого з'єднання також включає обробку помилок та відновлення з'єднання. Вона відстежує стан мережевого з'єднання, виявляє помилки підключення або втрату з'єднання та намагається відновити з'єднання автоматично або надає повідомлення користувачу про проблеми з мережею [56].

Усі ці функції спроектовані для забезпечення надійного та стабільного мережевого з'єднання між клієнтом та сервером месенджера, що дозволяє ефективно обмінюватися даними та забезпечувати безперебійну роботу месенджера для користувачів.

У додатку В представлений код клієнтської сторони мережевого з’єднання з Firebase на мові JavaScript.

Нижче наведено його покрокове пояснення:

1. Ми імпортуємо Firebase SDK та необхідні модулі для аутентифікації та роботи з Firebase Realtime Database.
2. Ми ініціалізуємо конфігурацію Firebase з даними користувача, включаючи API-ключ, домен авторизації, URL бази даних та інші параметри.
3. Ми ініціалізуємо з'єднання з Firebase, використовуючи ініціалізовану конфігурацію.
4. Ми отримуємо посилання на об'єкт бази даних Firebase для подальшої роботи з ним.
5. Ми здійснюємо аутентифікацію користувача, використовуючи метод signInWithEmailAndPassword() та передаючи облікові дані користувача (електронну пошту та пароль).
6. Якщо аутентифікація успішна, ми можемо виконувати необхідні дії з Firebase Realtime Database.
7. У даному прикладі ми демонструємо дві дії: отримання даних з бази даних та відправлення даних до бази даних. Ми використовуємо методи once() та set() для цих операцій.
8. В разі успішного виконання дій ми обробляємо результати або виводимо повідомлення про помилку, якщо щось пішло не так.

3.10 Висновки

У третьому розділі було розглянуто різні аспекти реалізації месенджера, включаючи вибір технологій, архітектуру системи, компоненти, використання Firebase та особливості серверної та клієнтської частин месенджера. Розуміння цих аспектів допоможе успішно реалізувати месенджер з бажаною функціональністю та надійністю. Основні висновки з цього розділу наступні:

1. Обраний стек технологій дозволяє реалізувати месенджер з високою продуктивністю та надійністю. Він включає мови програмування, фреймворки та інструменти, які найкраще підходять для даного проекту.

2. Архітектура системи месенджера грає важливу роль у розподілі функцій та взаємодії між компонентами системи. Популярні архітектурні підходи, такі як MVC або MVVM, можуть бути використані для структурування клієнтської частини месенджера.

3. Компоненти системи месенджера включають серверну та клієнтську частини. Серверна частина відповідає за обробку логіки, збереження даних та керування мережевим з'єднанням. Клієнтська частина забезпечує графічний інтерфейс для користувача та взаємодію з сервером.

4. Firebase є потужним інструментом для реалізації месенджера. Вона надає рішення для аутентифікації користувачів, збереження даних в реальному часі та обробки повідомлень.

5. Важливо розробити серверну частину месенджера, яка буде забезпечувати безпеку, ефективність та надійність. Це включає захист даних, оптимізацію запитів до бази даних та забезпечення масштабованості.

6. Клієнтська частина месенджера повинна мати зручний та інтуїтивно зрозумілий графічний інтерфейс, який дозволить користувачам з легкістю взаємодіяти з месенджером. Також вона повинна забезпечувати зручний спосіб відправки, отримання та відображення повідомлень.

РОЗДІЛ 4
КОНФІДЕНЦІЙНИЙ ОБМІН ГОЛОСОВИМИ ПОВІДОМЛЕННЯМИ

4.1 Загальні положення

У сучасному цифровому світі, де комунікація здійснюється переважно через електронні канали, захист конфіденційності голосових повідомлень є вкрай важливим. У 2023 році, коли сфера онлайн комунікації є неодмінною частиною нашого повсякденного життя, голосові повідомлення стають дедалі більш поширеними способом спілкування.

Однак, разом зі зростанням популярності голосових повідомлень, збільшується й ризик несанкціонованого доступу до особистої інформації, яка міститься у цих повідомленнях. Завдяки технологічному прогресу, зловмисники можуть намагатись перехопити та розшифрувати голосові повідомлення, отримати доступ до особистих даних, таких як адреси, номери телефонів, підприємницьку інформацію та інше.

Забезпечення конфіденційності голосових повідомлень стає невід'ємною складовою сфери приватності та безпеки користувачів. Захищені голосові повідомлення дозволяють людям спілкуватися, обмінюватися інформацією та виконувати бізнес-операції, знаючи, що їх дані захищені від небажаних очей. Це сприяє збереженню особистої інтимності, довіри та забезпечує відчуття безпеки у віртуальному середовищі.

У деяких сферах, таких як медицина, фінанси, ділова комунікація та державна безпека, захист голосових повідомлень має особливу вагу. Вони містять конфіденційну інформацію, що може мати значення для життя та безпеки людей, а також впливати на економічні та політичні процеси. В цих випадках недостатня безпека голосових повідомлень може мати серйозні наслідки, включаючи виток конфіденційних даних, шкоду бізнесу, шпигунство та зловживання.

Таким чином, в 2023 році захист голосових повідомлень стає невід'ємною потребою, щоб забезпечити приватність, довіру та безпеку у цифровому світі.

Використання криптографічних методів шифрування, таких як генератори псевдовипадкових чисел та шифрування AES, разом з надійними сервісами збереження даних, такими як Firebase Database, дозволяє забезпечити надійний рівень конфіденційності голосових повідомлень.

AES (Advanced Encryption Standard) – це симетричний алгоритм шифрування, який використовується для захисту конфіденційності даних. Він є одним з найбільш поширених і надійних алгоритмів шифрування, які зараз використовуються.

AES базується на заміщенні та перестановці байтів у блоках даних, що шифруються, за допомогою спеціально підібраних ключів. Цей процес виконується у кілька раундів шифрування, де кожен раунд включає певні підстановки та перестановки байтів. Завдяки своїм властивостям шифрування та надійності, AES широко використовується для захисту конфіденційної інформації в різних сферах, включаючи фінанси, здоров'я, комунікації та багато інших [57].

4.2 Історія зламів голосових повідомлень: Виклики та боротьба з безпекою

Злам голосових повідомлень є одним з аспектів кібербезпеки, що займається незаконним доступом до аудіофайлів або голосових комунікацій. Історія зламів голосових повідомлень пов'язана з еволюцією технологій зв'язку та збільшенням кількості людей, що використовують голосові послуги.

У минулому, коли телефонна система була основним засобом спілкування, злам голосових повідомлень не був настільки поширеним явищем. Однак, з розвитком цифрових технологій та мобільних комунікацій, зловмисники стали використовувати різні методи для отримання доступу до голосових даних.

У 2000-х роках, з появою голосової пошти, зловмисники почали використовувати методи підміни або перехоплення голосових повідомлень. Це дозволяло їм отримувати доступ до повідомлень, що призводило до проблем з приватністю і безпекою особистих даних.

З впровадженням голосових асистентів, таких як Siri, Google Assistant та Amazon Alexa, з'явилися нові виклики щодо безпеки голосових повідомлень.

Зловмисники спробували експлуатувати вразливості систем штучного інтелекту, щоб отримати несанкціонований доступ до голосових команд і данних користувачів.

Одним з найбільш відомих прикладів зламу голосових повідомлень є справа з компанією Apple в 2014 році. У цьому випадку, хакери отримали незаконний доступ до голосових повідомлень з голосової пошти користувачів, використовуючи вразливості в безпеці системи.

Загальна історія зламів голосових повідомлень свідчить про постійну боротьбу між зловмисниками та розробниками систем комунікації. Розробники постійно вдосконалюють методи шифрування, встановлюють більш сильні механізми аутентифікації та розробляють строгі політики безпеки для захисту голосових повідомлень від несанкціонованого доступу.

Незважаючи на постійний розвиток технологій та зусилля розробників, злам голосових повідомлень залишається потенційною загрозою. Тому важливо дотримуватись найкращих практик з кібербезпеки, використовувати надійні шифрувальні алгоритми та регулярно оновлювати системи для запобігання зламам та збереження конфіденційності голосових повідомлень [58].

4.3 Переваги обраного рішення

Забезпечення конфіденційності голосових повідомлень за допомогою криптографічного генератора псевдовипадкових чисел, шифрування AES та Firebase Database є гарним рішенням з кількох причин:

1. Високий рівень безпеки: криптографічний генератор псевдовипадкових чисел гарантує створення непередбачуваних ключів шифрування, що робить зламування даних практично неможливим. AES (Advanced Encryption Standard) є сильним алгоритмом шифрування, який використовується у багатьох системах для забезпечення безпеки даних. Використання цих криптографічних методів гарантує, що голосові повідомлення залишаються конфіденційними і недоступними для несанкціонованого доступу.

2. Цілісність даних: шифрування AES забезпечує не лише конфіденційність, але й цілісність даних. Це означає, що будь-яка зміна або модифікація голосового повідомлення буде виявлена, оскільки будь-яке невідповідність у шифрованому тексті при розшифруванні буде помітна.

3. Легкість інтеграції: використання Firebase Database дозволяє зручно зберігати і обмінюватися зашифрованими голосовими повідомленнями. Firebase забезпечує безпечне зберігання даних в хмарному середовищі, що дозволяє легко забезпечити доступ до повідомлень на різних пристроях.

4. Переносимість та доступність: використання криптографічних методів та Firebase Database дозволяє забезпечити безпеку голосових повідомлень на різних платформах та пристроях. Це робить їх доступними та зручними для користувачів незалежно від їх пристрою чи операційної системи.

5. Довіра та репутація: забезпечення конфіденційності голосових повідомлень є важливим аспектом будь-якої комунікаційної системи. Це дозволяє користувачам довіряти платформі та почувати себе захищеними під час обміну особистою інформацією.

Загалом, використання криптографічного генератора псевдовипадкових чисел, шифрування AES та Firebase Database є ефективним рішенням для забезпечення конфіденційності голосових повідомлень у месенджерах. Це дозволяє користувачам комунікувати безпечно, зберігаючи їх приватні дані зашифрованими і захищеними від несанкціонованого доступу.

4.4 Потенційні недоліки

Хоча забезпечення конфіденційності голосових повідомлень за допомогою криптографічного генератора псевдовипадкових чисел, шифрування AES та Firebase Database має свої переваги, є деякі потенційні недоліки, які варто враховувати:

1. Залежність від сторонніх сервісів: використання Firebase Database означає, що надійність та безпека системи залежить від довіри до постачальника хмарних послуг. Якщо Firebase стикається зі своїми проблемами безпеки або

виникають проблеми з доступом до сервісу, це може вплинути на конфіденційність голосових повідомлень.

2. Вразливість до атак на рівень застосунку: хоча шифрування AES є сильним алгоритмом, сам процес шифрування і розшифрування залежить від правильної реалізації та безпеки самого застосунку. Якщо реалізація не є належною, можливість атак на рівень застосунку (наприклад, атака "людського фактору" або використання недостатньо захищених паролів) може компрометувати конфіденційність голосових повідомлень.

3. Можливість атак на передачу даних: хоча шифрування AES забезпечує конфіденційність даних в спокійних умовах зберігання, воно не надає захисту від атак на передачу даних. Наприклад, якщо злоумисник здобуде доступ до засекречених ключів або перехопить зашифровані повідомлення під час їх передачі, він може використати ці дані для несанкціонованого доступу до голосових повідомлень.

4. Важкість відновлення в разі втрати даних: якщо ключі шифрування або зашифровані дані втрачаються або пошкоджуються, може виникнути проблема зі відновленням доступу до голосових повідомлень. Важливо робити резервні копії ключів та даних, а також мати механізми відновлення для запобігання втраті даних [59].

4.5 Програмна реалізація

Цей підрозділ присвячений плану програмної реалізації та лістингу коду шифрування голосових повідомлень за допомогою шифрування AES, приватні ключі якого будуть згенеровані криптографічним генератором псевдовипадкових чисел. Після шифрування повідомлення буде надіслане до Firebase Database та передане стороні-отримувачу.

Процес поетапної реалізації шифрування голосових повідомлень на Kotlin з використанням генератора псевдовипадкових чисел для генерації ключів для AES наступний:

1. Імпортування необхідних бібліотек: перед початком реалізації шифрування, необхідно імпортувати необхідні бібліотеки для роботи з криптографічними алгоритмами та генераторами псевдовипадкових чисел. Наприклад, можна використати бібліотеку javax.crypto для роботи з AES і java.security.SecureRandom для генерації псевдовипадкових чисел.

2. Генерація ключів: використовуючи генератор псевдовипадкових чисел, можна згенерувати випадковий ключ для шифрування голосового повідомлення з використанням AES. Для цього створюється екземпляр класу SecureRandom, який генерує псевдовипадкові числа. Згенерований ключ потім використовується для ініціалізації об'єкту SecretKeySpec, який представляє ключ для AES.

3. Шифрування голосового повідомлення: за допомогою отриманого ключа можна зашифрувати голосове повідомлення за допомогою AES. Для цього створюється екземпляр класу Cipher з використанням алгоритму AES та режиму шифрування. Повідомлення перетворюється на байтовий масив і шифрується за допомогою Cipher.doFinal().

4. Розшифрування повідомлення: отримувач, який має доступ до ключа, може розшифрувати отримане зашифроване голосове повідомлення. Він створює екземпляр класу Cipher з використанням того ж ключа і алгоритму AES, але в режимі розшифрування. Зашифроване повідомлення розшифровується за допомогою Cipher.doFinal(), і результат перетворюється назад у вихідний формат повідомлення.

5. Збереження ключів: ключі шифрування, які використовуються для голосових повідомлень, повинні бути збережені безпечно. Це означає їх зберігання в захищеній базі даних Firebase Database.

6. Обробка помилок та винятків: важливо враховувати можливі помилки та винятки, які можуть виникнути під час процесу шифрування. Для цього можна використовувати конструкцію try-catch, щоб перехопити та обробити винятки, що можуть виникнути під час генерації ключів, шифрування або розшифрування.

На рисунках 4.1, 4.2 наведено блок-схеми, що ілюструють процес конфіденційного обміну голосовими повідомлення з боку клієнтської та серверної сторони розробленого програмного застосунку.

Рисунок 4.1 – Процес конфіденційного обміну голосовими повідомлення з боку клієнтської частини

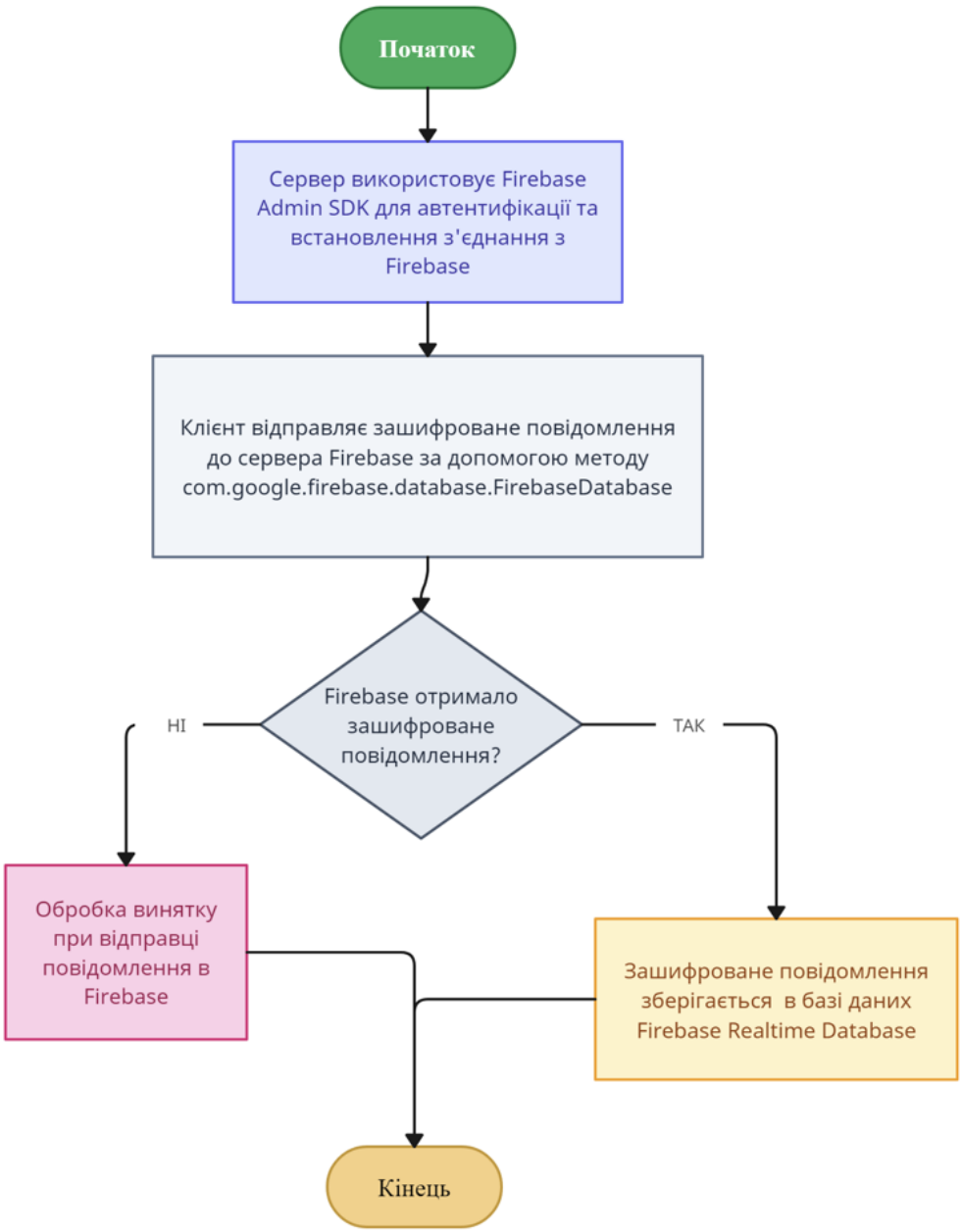

Рисунок 4.2 – Процес конфіденційного обміну голосовими повідомленнями з боку серверної частини

Нижче буде наведений використаний код на мові програмування kotlin для реалізації покращеного шифрування для голосових повідомлень в месенджері.

1. Імпорт необхідних класів і пакетів (рисунок 4.3):

- com.google.firebase.database.FirebaseDatabase є класом з бібліотеки Firebase, який надає функціональність для роботи з базою даних Firebase Realtime Database.

- javax.crypto.Cipher є класом з пакету javax.crypto, який надає функціональність для шифрування та розшифрування даних за допомогою різних алгоритмів шифрування.
- java.security.SecureRandom є класом з пакету java.security, який використовується для генерації випадкових значень в безпечному режимі.
- javax.crypto.KeyGenerator є класом з пакету javax.crypto, який використовується для генерації ключів для криптографічних алгоритмів.
- javax.crypto.spec.SecretKeySpec є класом з пакету javax.crypto.spec, який представляє секретний ключ для використання в алгоритмах шифрування.

```
import com.google.firebase.database.FirebaseDatabase
import javax.crypto.Cipher
import javax.crypto.KeyGenerator
import javax.crypto.spec.SecretKeySpec
import java.security.SecureRandom
```

Рисунок 4.3 – Імпорт необхідних класів і пакетів

2. Визначення функцій для роботи з шифруванням та дешифруванням повідомлень:

- generateAESKey(): генерує ключ AES, використовуючи KeyGenerator. У випадку помилки виникає виключна ситуація, яка відображається інструкцією e.printStackTrace(), і потім виняток перекидається для подальшої обробки (рисунок 4.4).

```
fun generateAESKey(): SecretKeySpec {
    try {
        val keyGenerator = KeyGenerator.getInstance("AES")
        keyGenerator.init(256, SecureRandom())
        val secretKey = keyGenerator.generateKey()
        return SecretKeySpec(secretKey.encoded, "AES")
    } catch (e: Exception) {
        // Обробка винятку при генерації ключа
        e.printStackTrace()
        throw e
    }
}
```

Рисунок 4.4 – Визначення функції generateAESKey()

- encryptVoiceMessage(message: ByteArray, secretKey: SecretKeySpec): зашифровує голосове повідомлення за допомогою шифру AES. Використовується Cipher для ініціалізації шифрування з вказаним ключем. Якщо виникає помилка, виняток відображається за допомогою e.printStackTrace() і перекидається (рисунок 4.5).

```
fun encryptVoiceMessage(message: ByteArray, secretKey: SecretKeySpec): ByteArray {
    try {
        val cipher = Cipher.getInstance("AES")
        cipher.init(Cipher.ENCRYPT_MODE, secretKey)
        return cipher.doFinal(message)
    } catch (e: Exception) {
        // Обробка винятку при шифруванні повідомлення
        e.printStackTrace()
        throw e
    }
}
```

Рисунок 4.5 – Визначення функції encryptVoiceMessage ()

- decryptVoiceMessage(encryptedMessage: ByteArray, secretKey: SecretKeySpec): розшифровує зашифроване повідомлення з використанням ключа AES. Використовується Cipher для ініціалізації розшифрування з вказаним ключем. У випадку помилки виняток відображається за допомогою e.printStackTrace() і перекидається (рисунок 4.6).

```
fun decryptVoiceMessage(encryptedMessage: ByteArray, secretKey: SecretKeySpec): ByteArray {
    try {
        val cipher = Cipher.getInstance("AES")
        cipher.init(Cipher.DECRYPT_MODE, secretKey)
        return cipher.doFinal(encryptedMessage)
    } catch (e: Exception) {
        // Обробка винятку при розшифруванні повідомлення
        e.printStackTrace()
        throw e
    }
}
```

Рисунок 4.6 – Визначення функції decryptVoiceMessage ()

3. Визначення функції sendEncryptedMessageToFirebase(encyptedMessage: ByteArray), яка відправляє зашифроване повідомлення в Firebase Realtime Database. Якщо виникає помилка при відправленні, виняток відображається за допомогою e.printStackTrace() і перекидається (рисунок 4.7).

```
fun sendEncryptedMessageToFirebase(encryptedMessage: ByteArray) {
    try {
        val database = FirebaseDatabase.getInstance()
        val reference = database.getReference("encrypted_messages")
        reference.setValue(encryptedMessage)
    } catch (e: Exception) {
        // Обробка винятку при відправці повідомлення в Firebase
        e.printStackTrace()
        throw e
    }
}
```

Рисунок 4.7 – Визначення функції sendEncryptedMessageToFirebase()

4. Основна функція main(), яка демонструє використання функцій (рисунок 4.8, 4.9):

- Створюється рядок voiceMessage, який містить голосове повідомлення для шифрування.
- Генерується ключ AES за допомогою generateAESKey().
- Голосове повідомлення шифрується за допомогою encryptVoiceMessage() з використанням згенерованого ключа AES.
- Зашифроване повідомлення відправляється в Firebase за допомогою sendEncryptedMessageToFirebase().
- У разі виникнення помилок під час будь-якого з цих кроків, виняток відображається за допомогою e.printStackTrace().

```
fun main() {
    try {
        val voiceMessage = "Голосове повідомлення".toByteArray()

        // Генеруємо ключ AES
        val secretKey = generateAESKey()

        // Шифруємо голосове повідомлення
        val encryptedMessage = encryptVoiceMessage(voiceMessage, secretKey)
```

Рисунок 4.8 – Функція main

```
        // Відправляємо зашифроване повідомлення в Firebase
        sendEncryptedMessageToFirebase(encryptedMessage)

        println("Зашифроване повідомлення було відправлене в Firebase.")
    } catch (e: Exception) {
        // Обробка винятку в головному блоку коду
        e.printStackTrace()
    }
}
```

Рисунок 4.9 – Функція main (продовження)

4.6 Висновки

У розділі 4 ми розглянули процес конфіденційного обміну голосовими повідомленнями за допомогою шифрування. Основні висновки даного розділу:

1. Для забезпечення конфіденційності голосових повідомлень використовується шифрування за допомогою алгоритму AES (Advanced Encryption Standard).

2. Для генерації секретного ключа AES використовується KeyGenerator з встановленням довжини ключа на 256 бітів. Для генерації випадкових чисел використовується SecureRandom.

3. Шифрування голосового повідомлення виконується за допомогою Cipher, де шифр ініціалізується у режимі шифрування (Cipher.ENCRYPT_MODE) з використанням секретного ключа.

4. Розшифрування голосового повідомлення виконується за допомогою Cipher, де шифр ініціалізується у режимі розшифрування (Cipher.DECRYPT_MODE) з використанням того самого секретного ключа.

5. Для зберігання та передачі зашифрованих повідомлень використовується база даних Firebase Realtime Database. Зашифроване повідомлення передається до Firebase за допомогою sendEncryptedMessageToFirebase(), де використовується FirebaseDatabase та його методи для взаємодії з базою даних.

6. У випадку виникнення помилок або винятків при генерації ключа, шифруванні, розшифруванні або відправці повідомлення в Firebase, використовується конструкція try-catch для обробки винятків та виведення відповідних повідомлень про помилки.

Загалом, розділ 4 демонструє приклад конфіденційного обміну голосовими повідомленнями з використанням шифрування AES та бази даних Firebase. Застосування шифрування дозволяє забезпечити конфіденційність та безпеку голосових повідомлень під час їх передачі та зберігання.

ВИСНОВКИ

У даній дипломній роботі було розроблено програмний застосунок для забезпечення конфіденційного обміну голосовими повідомленнями. Робота передбачала виконання кількох завдань, які були успішно виконані в процесі дослідження та розробки.

В процесі виконання роботи був проведений аналіз існуючих методів шифрування в системах обміну повідомленнями. В результаті аналізу було визначено, що для забезпечення конфіденційності голосових повідомлень найбільш підходящим методом є використання симетричного шифрування, зокрема алгоритму AES.

Були розроблені вимоги до програмного засобу з урахуванням конфіденційного обміну голосовими повідомленнями. Вимоги включали такі аспекти, як безпека шифрування, аутентифікація користувачів, збереження даних та інтерфейс користувача.

Архітектура програмного застосунку була розроблена з урахуванням потреб безпечного обміну голосовими повідомленнями. Було використано клієнт-серверну модель, де серверна частина виконує функції зберігання даних та обробки запитів, а клієнтські додатки забезпечують комунікацію з користувачами.

В результаті розробки був отриманий програмний код застосунку, який включає реалізацію шифрування голосових повідомлень за допомогою AES, механізм аутентифікації користувачів та інші функції, необхідні для безпечного обміну повідомленнями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Evolution of Messaging Apps: A Brief Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.oberlo.com/blog/messaging-apps.

2. Ensuring Confidential Voice Messaging: Challenges and Solutions [Електронний ресурс] – Режим доступу до ресурсу: https://www.owasp.com/confidential-voice-messaging.

3. Centralized vs Decentralized Messaging Apps: Pros and Cons [Електронний ресурс] – Режим доступу до ресурсу: https://www.mitre.com/centralized-vs-decentralized-messaging-apps.

4. Shostack A. Threat Modeling: Designing for Security / Adam Shostack. Indianapolis: John Wiley & Sons, Inc., 2014. – 624 с.

5. The STRIDE per Element Chart [Електронний ресурс] – 2007. – Режим доступу до ресурсу: https://www.microsoft.com/security/blog/2007/10/29/the-stride-per-element-chart/.

6. End-to-End Encryption in Messaging Apps: Why It Matters [Електронний ресурс] –. Режим доступу до ресурсу: https://www.eff.org/secure-messaging.

7. . The Importance of Privacy in Messaging Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.wired.com/privacy-messaging-apps.

8. Secure Messaging: Protecting Your Conversations [Електронний ресурс] – Режим доступу до ресурсу: https://www.cnet.com/secure-messaging.

9. Confidentiality and Data Security in Messaging Platforms [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/confidentiality-data-security-messaging.

10. Ensuring Privacy: Best Practices for Messaging App Developers [Електронний ресурс] – Режим доступу до ресурсу: https://www.privacytools.io/privacy-best-practices-messaging-apps.

11. The Role of Encryption in Secure Messaging= [Електронний ресурс] –Режим доступу до ресурсу: https://www.digitaltrends.com/encryption-secure-messaging.

12. Privacy and Security Features of Popular Messaging Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.techradar.com/privacy-security-messaging-apps.

13. Messaging App Security: Protecting Your Data [Електронний ресурс] – Режим доступу до ресурсу: https://www.pcmag.com/messaging-app-security.

14. End-to-End Encryption: The Key to Messaging Privacy [Електронний ресурс] – Режим доступу до ресурсу: https://www.zdnet.com/end-to-end-encryption-messaging-privacy.

15. Privacy and Confidentiality in Messaging Platforms [Електронний ресурс] – Режим доступу до ресурсу: https://www.acs.org.au/privacy-confidentiality-messaging-platforms.

16. Data Privacy in Messaging Apps: What You Need to Know [Електронний ресурс] – Режим доступу до ресурсу: https://www.privacyrights.org/data-privacy-messaging-apps.

17. Common Attack Pattern Enumeration and Classification – [Електронний ресурс] – Режим доступу до ресурсу: https://capec.mitre.org/index.html.

18. OWASP Top 10 [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-project-top-ten/.

19. Common Weakness Enumeration [Електронний ресурс] – Режим доступу до ресурсу: https://cwe.mitre.org/about/index.html.

20. Lautenbach A. HEAVENS [Електронний ресурс] / A. Lautenbach, M. Islam. – 2016. – Режим доступу до ресурсу: https://autosec.se/wp-content/uploads/2018/03/HEAVENS_D2_v2.0.pdf.

21. Surface vehicle recommended practice (J3061) [Електронний ресурс] – Режим доступу до ресурсу: https://www.sae.org/standards/content/j3061/.

22. Secure Messaging Apps: A Comprehensive Review [Електронний ресурс] – Режим доступу до ресурсу: https://www.privacytools.io/messaging/.

23. Privacy and Security in Instant Messaging Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/privacy-security-instant-messaging-apps.

24. The Importance of End-to-End Encryption in Messaging [Електронний ресурс] – Режим доступу до ресурсу: https://www.wired.com/importance-end-to-end-encryption-messaging.

25. Messaging Privacy: Best Practices for Secure Communication [Електронний ресурс] – Режим доступу до ресурсу: https://www.eff.org/messaging-privacy-best-practices.

26. Confidentiality in Messaging Platforms: Challenges and Solutions [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/confidentiality-messaging-platforms.

27. Secure Messaging: Protecting Your Privacy Online [Електронний ресурс] – Режим доступу до ресурсу: https://www.consumer.ftc.gov/secure-messaging-privacy.

28. Introduction to Modern Cryptography [Електронний ресурс] – Режим доступу до ресурсу: https://web.cs.ucdavis.edu/~rogaway/classes/227/spring05/intro.pdf.

29. Privacy and Security Features of Popular Messaging Apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.oberlo.com/privacy-security-features-messaging-apps.

30. Messaging App Security: Best Practices for Protecting Your Data [Електронний ресурс] – Режим доступу до ресурсу: https://www.techrepublic.com/messaging-app-security-best-practices.

31. The Importance of End-to-End Encryption in Messaging Apps [Електронний ресурс] - Режим доступу до ресурсу: https://www.securityintelligence.com/posts/the-importance-of-end-to-end-encryption-in-messaging-apps/.

32. Securing Your Messaging App: Best Practices for Privacy and Security [Електронний ресурс] - Режим доступу до ресурсу: https://www.synopsys.com/blogs/software-security/securing-messaging-app-best-practices/.

33. Privacy and Security Features in Popular Messaging Apps [Електронний ресурс] - Режим доступу до ресурсу: https://www.wired.com/story/secure-messaging-apps-privacy-features/.

34. Messaging App Security: How to Protect Your Privacy [Електронний ресурс] - Режим доступу до ресурсу: https://heimdalsecurity.com/blog/messaging-app-security-protect-privacy/.

35. The Role of Encryption in Secure Messaging Apps [Електронний ресурс] - Режим доступу до ресурсу: https://www.eff.org/wp/encrypting-messenger.

36. Secure Messaging Apps: A Guide to Choosing the Right One for You [Електронний ресурс] - Режим доступу до ресурсу: https://www.tomsguide.com/us/secure-messaging-apps,review-3002.html.

37. The Best Encrypted Messaging Apps You Should Use Today [Електронний ресурс] - Режим доступу до ресурсу: https://www.techradar.com/best/encrypted-messaging-app.

38. Secure Messaging Apps Comparison Chart [Електронний ресурс] - Режим доступу до ресурсу: https://www.privacytools.io/software/real-time-communication/.

39. Privacy and Security in Instant Messaging Apps: A Systematic Review [Електронний ресурс] - Режим доступу до ресурсу: https://www.researchgate.net/publication/336963011_Privacy_and_Security_in_Instant_Messaging_Apps_A_Systematic_Review.

40. Messaging App Security: A Comprehensive Guide [Електронний ресурс] - Режим доступу до ресурсу: https://www.checkmarx.com/blog/messaging-app-security-comprehensive-guide/.

41. The Ultimate Guide to Messaging App Security [Електронний ресурс] - Режим доступу до ресурсу: https://www.varonis.com/blog/messaging-app-security/.

42. Secure Messaging Apps: A Comprehensive Guide [Електронний ресурс] - Режим доступу до ресурсу: https://www.mcafee.com/blogs/consumer/secure-messaging-apps/.

43. The Importance of End-to-End Encryption in Messaging Apps [Електронний ресурс] - Режим доступу до ресурсу: https://protonmail.com/blog/end-to-end-encryption-messaging-apps/.

44. Messaging App Security: Protecting Your Privacy [Електронний ресурс] - Режим доступу до ресурсу: https://www.avast.com/c-messaging-app-security.

45. Best Encrypted Messaging Apps for Android & iOS [Електронний ресурс] - Режим доступу до ресурсу: https://www.expressvpn.com/blog/best-encrypted-messaging-apps/.

46. Secure Messaging: The Key to Digital Privacy [Електронний ресурс] - Режим доступу до ресурсу: https://www.wired.com/story/secure-messaging-guide/.

47. How to Choose a Secure Messaging App [Електронний ресурс] - Режим доступу до ресурсу: https://ssd.eff.org/en/module/how-choose-messenger.

48. Privacy and Security in Instant Messaging [Електронний ресурс] - Режим доступу до ресурсу: https://www.schneier.com/essays/archives/2008/01/privacy_and_security_1.html.

49. Messaging App Security: What You Need to Know [Електронний ресурс] - Режим доступу до ресурсу: https://www.tomsguide.com/round-up/secure-messaging-apps.

50. The Rise of Encrypted Messaging Apps: Benefits and Concerns [Електронний ресурс] - Режим доступу до ресурсу: https://www.comparitech.com/blog/information-security/encrypted-messaging-apps-benefits-concerns/.

51. The Importance of Privacy in Messaging Apps [Електронний ресурс] - Режим доступу до ресурсу: https://www.digitaltrends.com/mobile/privacy-in-messaging-apps/.

52. Messaging App Security: What You Should Know [Електронний ресурс] - Режим доступу до ресурсу: https://www.intego.com/mac-security-blog/messaging-app-security-what-you-should-know/.

53. Secure Messaging Apps: A Guide to Choosing the Best Option [Електронний ресурс] - Режим доступу до ресурсу: https://securitytrails.com/blog/secure-messaging-apps.

54. How Secure Are Messaging Apps? A Comprehensive Guide [Електронний ресурс] - Режим доступу до ресурсу: https://www.comparitech.com/blog/information-security/how-secure-are-messaging-apps/.

55. End-to-End Encryption in Messaging Apps: Benefits and Challenges [Електронний ресурс] - Режим доступу до ресурсу: https://www.mcafee.com/blogs/consumer/end-to-end-encryption-messaging-apps/.

56. Understanding Encryption: A Guide to the Basics [Електронний ресурс] - Режим доступу до ресурсу: https://www.varonis.com/blog/what-is-encryption/.

57. End-to-End Encryption: What It Is and Why It Matters [Електронний ресурс] - Режим доступу до ресурсу: https://www.wired.com/story/end-to-end-encryption/.

58. How Encryption Works in Messaging Apps [Електронний ресурс] - Режим доступу до ресурсу: https://securityintelligence.com/how-does-encryption-work-in-messaging-apps/.

59. The Importance of Encryption in Data Security [Електронний ресурс] - Режим доступу до ресурсу: https://www.techradar.com/news/the-importance-of-encryption-in-data-security.

ДОДАТОК А

Лістинг коду на мові програмування Node.js для розробки серверної частини месенджера зі зв'язком до Firebase та клієнтською частиною:

```
//Імпорт необхідного модуля firebase-admin і налаштування з'єднання з Firebase
const admin = require('firebase-admin');
const serviceAccount = require('./path/to/serviceAccountKey.json');
admin.initializeApp({
  credential: admin.credential.cert(serviceAccount),
  databaseURL: 'https://project.firebaseio.com'
});
//Отримання посилання на базу даних Firebase
const db = admin.database();
//Написання функції для збереження даних у базі даних Firebase
function saveDataToFirebase(data) {
  const ref = db.ref('collectionName');
  ref.push(data)
    .then(() => {
      console.log('Дані успішно збережені у Firebase');
    })
    .catch((error) => {
      console.error('Помилка при збереженні даних у Firebase:', error);
    });
}
//Написання функції для отримання даних з бази даних Firebase:
function getDataFromFirebase() {
  const ref = db.ref('collectionName');
  ref.once('value')
    .then((snapshot) => {
```

```
      const data = snapshot.val();
      console.log('Отримані дані з Firebase:', data);
    })
    .catch((error) => {
      console.error('Помилка при отриманні даних з Firebase:', error);
    });
}
```

1. Виклик функцій для взаємодії з Firebase:

```
saveDataToFirebase({ name: 'Gleb', age: 23 });
getDataFromFirebase();
```

ДОДАТОК Б

Лістинг коду для екрану розмови клієнтської частини месенджера:

```
// Оголошення класу для екрану розмови
class ConversationActivity : AppCompatActivity() {
    // Ініціалізація змінних та компонентів екрану
    private lateinit var recyclerView: RecyclerView
    private lateinit var messageAdapter: MessageAdapter
    private lateinit var messageList: MutableList<Message>
    private lateinit var sendMessageButton: Button
    private lateinit var messageInput: EditText
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_conversation)
        // Знаходження та ініціалізація компонентів екрану
        recyclerView = findViewById(R.id.messageRecyclerView)
        sendMessageButton = findViewById(R.id.sendMessageButton)
        messageInput = findViewById(R.id.messageInput)
        // Ініціалізація списку повідомлень та адаптера
        messageList = mutableListOf()
        messageAdapter = MessageAdapter(messageList)
        // Налаштування RecyclerView для відображення повідомлень
        recyclerView.apply {
            layoutManager = LinearLayoutManager(context)
            adapter = messageAdapter
        }
        // Додавання обробника події для кнопки надсилання повідомлення
        sendMessageButton.setOnClickListener {
            val messageText = messageInput.text.toString()
```

```
            if (messageText.isNotEmpty()) {
                // Створення об'єкту повідомлення та додавання його до списку
                val message = Message(messageText)
                messageList.add(message)
                messageAdapter.notifyItemInserted(messageList.size - 1)
                // Очищення поля вводу повідомлення
                messageInput.text.clear()
            }
        }
    }
}
```

ДОДАТОК В

Лістинг коду клієнтської сторони мережевого з’єднання з Firebase на мові JavaScript:

```
// Імпортуємо Firebase SDK та необхідні модулі
const firebase = require('firebase/app');
require('firebase/auth');
require('firebase/database');
// Ініціалізуємо конфігурацію Firebase з вашими налаштуваннями
const firebaseConfig = {
  apiKey: 'YOUR_API_KEY',
  authDomain: 'YOUR_AUTH_DOMAIN',
  databaseURL: 'YOUR_DATABASE_URL',
  projectId: 'YOUR_PROJECT_ID',
  storageBucket: 'YOUR_STORAGE_BUCKET',
  messagingSenderId: 'YOUR_SENDER_ID',
  appId: 'YOUR_APP_ID',
};
// Ініціалізуємо з'єднання з Firebase
firebase.initializeApp(firebaseConfig);
// Отримуємо посилання на об'єкт бази даних
const database = firebase.database();
// Здійснюємо аутентифікацію користувача
firebase.auth().signInWithEmailAndPassword('email@example.com', 'password')
  .then(() => {
    // Отримання даних з бази даних
    database.ref('messages').once('value')
      .then((snapshot) => {
        const messages = snapshot.val();
```

```
      // Обробляємо отримані повідомлення
      console.log(messages);
    })
    .catch((error) => {
      console.error('Помилка при отриманні даних:', error);
    });
  // Відправлення даних до бази даних
  const newMessageRef = database.ref('messages').push();
  newMessageRef.set({
    text: 'Привіт, світ!',
    timestamp: Date.now(),
  })
    .then(() => {
      console.log('Повідомлення успішно додано до бази даних');
    })
    .catch((error) => {
      console.error('Помилка при додаванні повідомлення:', error);
    });
})
.catch((error) => {
  console.error('Помилка авторизації:', error);
});
```