Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

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

на тему СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ОПРАЦЮВАННЯ ДАНИХ ДЛЯ СПІЛКУВАННЯ У ЛОКАЛЬНІЙ МЕРЕЖІ
DESIGN OF A DATA PROCESSING SYSTEM FOR COMMUNICATION IN A LOCAL NETWORK

Виконав(ла): студент(ка) 4 курсу, групи КНТ-213сп
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Комп'ютерні науки

КУЛИКОВ В.А.
(ПРІЗВИЩЕ та ініціали)
Керівник КОЛПАКОВА Т.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОЦУР М.І.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КУЛИКОВА Віталія Андрійовича
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Створення програмної системи опрацювання даних для спілкування у локальній мережі. Design of a Data Processing System for Communication in a Local Network

керівник проєкту (роботи) к.т.н., доцент, КОЛПАКОВА Тетяна Олексіївна ,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від " 7 " квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 03 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис системи опрацювання даних. 4. Експлуатація, тестування та експериментальне дослідження системи опрацювання даних.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КОЛПАКОВА Т.О., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 7 ” квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка структури системи опрацювання даних.	2 тиждень	Розділ 3
5	Розробка системи опрацювання даних.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	системи опрацювання даних.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Віталій КУЛИКОВ
(підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Тетяна КОЛПАКОВА
(підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 93 с., 5 табл., 25 рис., 3 дод., 22 джерела.

JAVASCRIPT, MYSQL, REACT, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЛОКАЛЬНА МЕРЕЖА, МОВА ПРОГРАМУВАННЯ, СИСТЕМА ОПРАЦЮВАННЯ ДАНИХ.

Об’єкт дослідження – процеси обчислень, пов’язані з організацією та опрацюванням даних для обміну інформацією у локальній мережі.

Предмет дослідження – програмні системи опрацювання даних для спілкування у локальній мережі.

Мета роботи – розробка програмної системи опрацювання даних для спілкування у локальній мережі для підвищення ефективності взаємодії користувачів.

Матеріали, методи та технічні засоби: мова програмування JavaScript, бібліотека для побудови інтерфейсів React, система управління базами даних MySQL.

Результати. Розроблено проєктні рішення для створення системи опрацювання даних для спілкування у локальній мережі. Створено систему опрацювання даних для спілкування у локальній мережі. Проведено дослідження розробленої системи опрацювання даних для спілкування у локальній мережі.

Висновки. Мету роботи досягнуто. Розроблено систему опрацювання даних для спілкування у локальній мережі за допомогою мови програмування JavaScript, бібліотеки для побудови інтерфейсів React та системи управління базами даних MySQL.

Галузь використання – спілкування користувачів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 93 pages, 5 tables, 25 figures, 3 appendixes, 22 sources.

JAVASCRIPT, MYSQL, REACT, DATABASE, WEB APPLICATION, LOCAL NETWORK, PROGRAMMING LANGUAGE, DATA PROCESSING SYSTEM.

The object of research is computing processes related to the organization and processing of data for information exchange in a local network.

The subject of the research is data processing systems for communication in a local network.

The purpose of this work is to develop the data processing system for communication in a local network to increase the efficiency of user interaction.

Materials, methods and technical tools: JavaScript programming language, React interface building library, MySQL database management system.

Results. Project solutions for the creation of data processing system for communication in a local network has been designed. The data processing system for communication in a local network has been developed. The study of the developed data processing system for communication in a local network has been conducted.

Conclusions. The goal of the work has been achieved. The data processing system for communication in a local network using the JavaScript programming language, the React interface building library, and the MySQL database management system has been developed.

Scope of use – user communication.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ІС – інформаційна система;

ЛМ – локальна мережа;

СОД – система опрацювання даних.

ВСТУП

Актуальність створення системи опрацювання даних для спілкування у локальній мережі зумовлюється потребою у надійному, швидкому та захищеному обміні інформацією всередині організації. Така система забезпечує підвищений рівень безпеки, оскільки передавання даних відбувається без виходу у зовнішні мережі, що мінімізує ризики перехоплення чи несанкціонованого доступу. Вона вирізняється високою швидкістю та стабільністю комунікацій завдяки пропускній здатності локальної інфраструктури, дозволяючи оперативно координувати спільну роботу користувачів [1], [2].

Локальна система комунікації не залежить від доступу до Інтернету та забезпечує безперервний обмін повідомленнями й файлами навіть за відсутності зовнішнього зв’язку. Її можна легко інтегрувати з іншими внутрішніми сервісами, що спрощує організацію командної взаємодії та сприяє підвищенню ефективності бізнес-процесів. Крім того, повний контроль над правами доступу дає змогу гарантувати конфіденційність інформації, здійснювати моніторинг активності та впроваджувати політики безпеки відповідно до вимог установи [3], [4].

Важливо й те, що розробка такої системи дає можливість налаштувати її під конкретні потреби організації, на відміну від універсальних засобів онлайн-комунікацій. Це дозволяє оптимізувати використання фінансових ресурсів та застосувати власну інфраструктуру без додаткових витрат на сторонні сервіси. Таким чином, розробка та впровадження системи опрацювання даних для спілкування у локальній мережі є актуальним кроком для підвищення безпеки, автономності, надійності та ефективності внутрішньої комунікації [1]-[7].

Проте деякі системи опрацювання даних для спілкування у локальній мережі є досить дорогими, крім того їх використання пов’язано з потребою в постійному технічному обслуговуванні мережевої інфраструктури та

серверного обладнання. Відмова будь-якого елемента або збій в енергопостачанні можуть призвести до повної зупинки комунікації. Забезпечення належного рівня безпеки також вимагає професійного адміністрування, оскільки локальні системи, без оновлень і моніторингу, можуть стати вразливими до внутрішніх загроз. Крім того, масштабування такої системи опрацювання даних при зростанні кількості користувачів або географічному розширенні організації може потребувати значних матеріальних і часових витрат [1]-[7].

Тому актуальною є розробка системи опрацювання даних для спілкування у локальній мережі. У дипломній кваліфікаційній роботі бакалавра розв’язується актуальне завдання розробки програмної системи опрацювання даних для спілкування у локальній мережі для підвищення ефективності взаємодії користувачів.

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв’язати такі задачі:

– виконати аналіз предметної області та систем опрацювання даних для спілкування у локальній мережі;

– здійснити проєктування системи опрацювання даних для спілкування у локальній мережі;

– створити систему опрацювання даних для спілкування у локальній мережі;

– дослідити розроблену систему опрацювання даних для спілкування у локальній мережі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Системи опрацювання даних для спілкування у локальній мережі

Система опрацювання даних для спілкування у локальній мережі забезпечує обмін повідомленнями та файлами між користувачами, під’єднаними до однієї мережі без використання зовнішніх серверів чи доступу до Інтернету. Така система обробляє, передає та зберігає дані у межах внутрішньої мережевої інфраструктури, забезпечуючи швидку та захищену комунікацію між пристроями [1]-[3].

Такі системи створюються для забезпечення безпосередньої взаємодії між користувачами, які знаходяться в одному офісі, будівлі або іншому локальному середовищі. Її використання дозволяє організувати групові та приватні чати, передавати інформацію без затримок, контролювати доступ до даних та гарантувати конфіденційність завдяки відсутності сторонніх вузлів у процесі обміну [1]-[3].

Системи опрацювання даних для спілкування у локальній мережі дозволяють забезпечити не тільки синхронне, а й асинхронне спілкування. У сучасних умовах, коли робота на відстані стала звичним явищем, особливого значення набуває взаємодія, що не вимагає одночасної присутності учасників. Такий спосіб обміну інформацією дозволяє людям підтримувати контакт з будь-якого місця та реагувати тоді, коли це зручно. Обговорення ролі подібного формату взаємодії активно триває, адже саме він визначає, як організовується співпраця в умовах цифрового середовища [1].

Асинхронне спілкування являє собою обмін думками і повідомленнями між кількома особами без потреби синхронності дій. Учасники надсилають інформацію незалежно від того, чи знаходяться вони поруч, і не чекають миттєвої реакції. Це може бути електронне листування, текстові повідомлення у застосунках, взаємодія у чатах або на форумах. У таких випадках відповідь залежить лише від зручності і часу співрозмовника [1].

На противагу цьому, синхронна комунікація потребує одночасної участі всіх, хто спілкується. Звичайна бесіда в реальному житті або розмова через відеозв’язок відбувається у режимі теперішнього моменту, тому вимагає повної присутності кожної сторони [1].

Вибір виду спілкування залежить від обставин. Якщо необхідно донести новини, передати важливі повідомлення без нагальної потреби в обговоренні або обдумати складні питання, що потребують ретельного аналізу, перевага зазвичай надається спілкуванню без прив’язки до часу. Також подібний формат часто застосовується для надання об’єктивного та продуманого зворотного зв’язку. Водночас ситуації, що вимагають швидкої реакції, активного обміну ідеями, проведення ділових зустрічей чи командної взаємодії в реальному часі, більше відповідають синхронному підходу [1].

Асинхронний формат має низку переваг. Він дозволяє продовжувати роботу над проєктами, навіть якщо члени команди перебувають у різних часових зонах або мають різний робочий графік. Завдяки цьому знижується залежність від одночасної доступності всіх учасників, що робить процес організації значно простішим. Крім того, він спрощує концентрацію на завданнях, дозволяючи самостійно обирати час для оброблення повідомлень та уникати зайвих переривань робочого процесу [1].

Попри переваги, подібне спілкування має й свої слабкі сторони. Зокрема, через відсутність невербальних сигналів іноді стає складно правильно зрозуміти емоційне забарвлення повідомлення. Інформація може сприйматися надто формальною або сухою, хоча насправді цього не передбачалося. Також взаємодія може втрачати елемент безпосереднього людського контакту, що інколи негативно впливає на командний дух [1].

Тому для ефективної організації роботи зазвичай застосовується поєднання різних підходів, що забезпечується завдяки системам опрацювання даних для спілкування у локальній мережі. Там, де важлива швидкість і живе спілкування, доречним є формат у реальному часі, а в інших випадках

зручніше використовувати можливості асинхронного обміну інформацією, що забезпечує гнучкість і комфорт кожного учасника співпраці [1], [2].

Месенджер являє собою систему опрацювання даних для спілкування, що забезпечує передачу повідомлень і мультимедійних даних між користувачами з майже миттєвим отриманням відповіді. Розвиток таких програмних систем відбувався поступово: від простих текстових рішень до багатофункціональних платформ, які нині поєднують різні способи комунікації, включаючи голосову та відеовзаємодію, синхронізацію файлів і використання сервісів, що підтримують робочі процеси [2], [3].

Серед перших програм, що дали старт сучасним месенджерам, відзначалися інструменти, які створювали можливість швидкого текстового діалогу через мережу. Згодом функціональність таких систем збільшувалася, і особливо помітним цей процес став із поширенням мобільних пристроїв, після чого засоби обміну повідомленнями стали невід’ємним компонентом цифрового повсякдення [2], [3].

Сучасні рішення у сфері обміну даними надають більше, ніж просто передачу тексту. Вони перетворилися на універсальні центри цифрової взаємодії, які об’єднують робочі та особисті комунікаційні потреби людини. Їхня функціональність охоплює можливість спільного обговорення в групах, підтримку шифрування для захисту приватності, використання ботів для автоматизації завдань, а також роботу на різних операційних системах з узгодженням інформації між пристроями [2], [3].

Роль таких систем помітна не лише в особистому спілкуванні. Вони створюють нові можливості для бізнесу, полегшуючи підтримку користувачів і взаємодію з аудиторією. Компаніям пропонується прямий канал комунікації, у якому можна відповідати на запити, інформувати про продукти, надавати рекомендації та проводити транзакції. Використання автоматизованих сценаріїв спрощує обробку звернень і дає змогу підтримувати контакт із великою кількістю людей без затримок [2], [3].

Месенджер може виступати й як інструмент електронної торгівлі, дозволяючи просувати товари, приймати оплату та супроводжувати клієнта протягом усього процесу покупки, не виходячи з діалогу. Такі особливості роблять його ключовим елементом сучасних цифрових сервісів і важливою ланкою у взаємодії між людьми незалежно від їхнього місця перебування [2], [3].

У діловому середовищі застосування месенджерів потребує продуманого підходу, адже від нього залежить, наскільки результативно буде організована взаємодія з клієнтами та партнерами. Використання таких інструментів передбачає формування чіткої політики комунікації, у якій визначається, з ким і з якою метою ведеться листування, як оцінюються підсумки спілкування та які принципи побудови повідомлень підтримують імідж організації. Важливим аспектом є встановлення правильного балансу між автоматизованими відповідями та участю співробітників, адже саме людський фактор забезпечує потрібний рівень довіри, особливо коли мова йде про складні запити. Успішна робота з інформацією вимагає також дотримання вимог щодо захисту персональних даних і постійного аналізу того, як користувачі реагують на комунікацію. Взаємодія в месенджерах стає ще ефективнішою, якщо її гармонійно поєднувати з іншими каналами зв’язку, зокрема з електронною поштою та соціальними мережами [2], [3].

Технологічні зміни постійно впливають на розвиток платформ обміну повідомленнями, і тому бізнес має адаптуватися до нових можливостей. Штучний інтелект поступово укріплює позиції в месенджерах, підтримуючи функції автогенерації відповідей, інтелектуальний переклад та рекомендації, що суттєво спрощує масштабування взаємодії з аудиторією. Доповнена реальність відкриває додаткові інструменти для презентації товарів і підсилення маркетингових кампаній через візуальний ефект присутності. Паралельно з цим підвищується увага до конфіденційності: платформи оснащуються механізмами шифрування, тимчасового збереження повідомлень та індивідуальними параметрами керування інформацією. Крім того,

месенджери починають відігравати роль комерційних майданчиків, де можливо здійснювати платежі та завершувати покупки без переходу до сторонніх сервісів [2], [3].

З огляду на такі тенденції, компаніям важливо стежити за розвитком цифрових інструментів і гнучко перебудовувати підходи до комунікації, щоб залишатися конкурентоспроможними та підтримувати ефективний діалог із користувачами в сучасному мережевому просторі [2], [3].

Чат-сервіси у локальних або глобальних мережах формують основу сучасного цифрового спілкування, оскільки забезпечують оперативний обмін інформацією як у побуті, так і під час професійної діяльності. Їх призначення полягає у створенні умов, за яких дані можуть передаватися майже миттєво, а користувачі підтримують безперервний діалог без значних затримок. Завдяки використанню технологій із мінімальною латентністю вдається досягати максимально природної комунікації навіть тоді, коли учасники перебувають у різних місцях [3], [4].

Функціонування таких рішень базується на постійному інформаційному обміні між клієнтською частиною і сервером, що дозволяє не лише доставляти повідомлення, а й відображати статус активності співрозмовників, синхронізувати вкладення, зберігати історію діалогів та підтримувати доступ до неї. Архітектура, що зазвичай складається з інтерфейсу користувача, сервера передавання та маршрутизації даних і системи збереження відомостей, дає змогу організувати повноцінний процес комунікації без розривів [3], [4].

Подібні системи часто надають можливість не тільки передавати текст, а й ділитися різними видами контенту, включаючи графіку, аудіо, відео та документи. Передбачені додаткові інструменти — сигнали про надходження нових повідомлень, відображення введення тексту співрозмовником, варіанти персоналізації інтерфейсу. У багатьох рішеннях реалізовано механізми, орієнтовані на користувачів з особливими потребами, що сприяє доступності цифрового спілкування [3], [4].

Розвиток технологій сприяв появі спеціалізованих функцій залежно від призначення програмного забезпечення. Одні системи оптимізовані під командну роботу, інші — зосереджені на соціальній взаємодії чи підтримці мультимедійних можливостей. Завдяки цьому інструменти обміну даними в реальному часі активно інтегруються у повсякденні процеси, підвищуючи зручність співпраці, підтримуючи швидке прийняття рішень і залишаючись ключовим компонентом сучасних інформаційних середовищ [3], [4].

Обмін даними в режимі миттєвої взаємодії став ключовим елементом у функціонуванні сучасних цифрових сервісів, охоплюючи найрізноманітніші сфери діяльності. У робочому середовищі такі інструменти забезпечують швидку координацію дій під час виконання спільних завдань і дозволяють підтримувати постійний контакт між членами команди незалежно від їхнього фактичного розташування. У повсякденному житті вони слугують засобом соціальної комунікації, значно спрощуючи спілкування між користувачами та сприяючи налагодженню нових зв’язків [3], [4].

Завдяки можливості інтегрувати інтелектуальні моделі з такими платформами з’являється новий рівень автоматизованої взаємодії. Алгоритми штучного інтелекту в реальному часі здатні обробляти стандартні звернення та надавати відповідь без участі оператора, тим самим прискорюючи обслуговування і зменшуючи навантаження на персонал. Безперервний доступ до сервісу підвищує його цінність і покращує враження користувачів [4], [5].

Застосування технологій миттєвого обміну інформацією стає важливим і для стрімінгових платформ: можливість одразу реагувати на події, коментувати та взаємодіяти з іншими глядачами формує ефект присутності та підвищує зацікавленість аудиторії. Паралельно у засобах відеоконференцзв’язку чат виконує роль додаткового каналу, що дозволяє обмінюватися повідомленнями без переривання основного спілкування [4], [5].

При створенні таких систем розробникам доводиться визначатися між повною власною розробкою і використанням готових сервісів інтеграції.

Самостійне створення компонентів забезпечує максимальний контроль над функціональністю, проте потребує значних ресурсів і високого рівня технічної складності. Доступ до спеціалізованих API дає змогу пришвидшити впровадження та зменшити витрати, отримавши водночас стабільну і масштабовану інфраструктуру [4], [5].

Основою технічної реалізації є безперервний двосторонній зв’язок між серверною частиною та клієнтськими пристроями, який зазвичай підтримується через спеціальні з’єднання. Задля забезпечення продуктивної роботи системи додатково використовуються інструменти кешування, системи обробки черг та надійні бази даних, що відповідають за збереження і швидкий доступ до інформації. Розміщення сервісів у хмарних середовищах дозволяє збільшувати ресурси при зростанні навантаження та гарантувати стабільність обміну повідомленнями [4], [5].

Таким чином, системи для обміну даними в реальному часі формують гнучке та технологічно розвинене середовище комунікації, яке продовжує вдосконалюватися і розширювати сфери застосування [4], [5].

Під час створення програмних рішень для оперативного обміну інформацією важливо не обмежуватися лише технічною можливістю передавання повідомлень. Відчуття комфорту під час використання, легкість орієнтації в інтерфейсі та жвава взаємодія з елементами застосунку безпосередньо впливають на те, як користувач сприймає сервіс і наскільки охоче повертається до нього. Тому велике значення має продуманий зовнішній вигляд, у якому екрани організовані таким чином, щоб будь-яка дія була очевидною, а доступ до основних можливостей не вимагав додаткових зусиль [5], [6].

Окрему увагу варто приділяти елементам, що передають динаміку діалогу, адже саме вони створюють ілюзію живого спілкування. Візуальні сигнали на кшталт індикатора набору тексту, позначок доставки, реакцій чи дрібної анімації допомагають краще орієнтуватися в перебігу розмови й швидше реагувати на події. Використання виразних медіаелементів, таких як

різноманітні зображення або декоративні символи, допомагає створити емоційнішу комунікацію та дозволяє адаптувати застосунок під вподобання аудиторії [5], [6].

Важливим чинником залишається швидкість обміну даними: очікується, що повідомлення надходитимуть без помітних затримок і система коректно працюватиме навіть тоді, коли кількість активних користувачів значно зростає. Саме тому використання сучасних технологій підтримки постійного з’єднання, локального кешування та оптимізованої доставки інформації стає необхідною передумовою для стабільної роботи. Корисною може бути і можливість переглядати історію обміну повідомленнями тоді, коли зв’язок тимчасово недоступний [5], [6].

Оскільки такі рішення оперують приватними даними, питання захисту інформації виходить на перший план. Надійні механізми шифрування, своєчасне виявлення ризиків і прозорі правила доступу до інформації забезпечують довіру користувачів. Крім того, їм має бути надано право самостійно визначати, якими саме відомостями ділитися, а також вільно керувати своєю цифровою ідентичністю — від обмеження контакту з небажаними співрозмовниками до повного видалення власного профілю. Такі підходи створюють умови для безпечної та приємної взаємодії в цифровому середовищі [5], [6].

Системи для миттєвого обміну повідомленнями в локальних мережах розглядаються як інтерактивне рішення, що створює швидкий канал взаємодії між користувачем і службою підтримки або іншими учасниками мережного середовища. Подібні програмні платформи виступають інструментом забезпечення комунікації без затримок, що дозволяє оперативно передавати інформацію та підтримувати постійний діалог у цифровому просторі [5], [6].

З часом технології оброблення таких повідомлень зазнали значної модернізації: від елементарних текстових чатів, орієнтованих лише на обмін короткими репліками, вони перейшли до функціонально насичених систем, які забезпечують гнучку взаємодію з користувачем, інтеграцію з аналітичними

сервісами, автоматизованими помічниками та інструментами керування клієнтськими даними [5], [6].

У сучасному середовищі цифрових послуг така форма комунікації стала базовою складовою організації обслуговування. Від користувачів очікується отримання відповідей без тривалого очікування, тому миттєва реакція на звернення вважається необхідною умовою якісної взаємодії. Платформа чату дозволяє вирішувати завдання без перемикання між додатками чи каналами зв’язку, що підвищує комфорт та пришвидшує отримання необхідної допомоги [6], [7].

Використання подібних рішень позитивно впливає на ефективність взаємодії: звернення обробляються оперативніше, що формує відчуття надійності та заохочує клієнта повертатися по послуги повторно. Таке середовище сприяє зменшенню ризику переривання комунікації, що особливо важливо у випадках, коли потрібно підтримати інтерес користувача до продукту або сервісу [6], [7].

Окрім підвищення рівня задоволеності, застосування технологій живого спілкування сприяє оптимізації ресурсів. Оператори мають можливість паралельно вести декілька діалогів, знижуючи навантаження на персонал і скорочуючи витрати на інші канали підтримки. У підсумку це створює збалансовану систему з високим рівнем доступності, що вигідна як для постачальників послуг, так і для їхніх користувачів [6], [7].

Технології онлайн спілкування в локальних мережах забезпечують не лише обмін повідомленнями, а й постійний потік даних, які одразу піддаються опрацюванню. Інформація, що надходить у процесі діалогу, використовується для визначення найбільш актуальних тем, типових труднощів користувачів та їхніх очікувань. Такі відомості стають важливим ресурсом для вдосконалення програмних рішень, оптимізації змісту вебресурсів і підвищення якості підтримки [6], [7].

Оцінювання ефективності подібних систем ґрунтується на аналізі швидкості реакції, результативності наданих консультацій та загального рівня

задоволеності відвідувачів. Завдяки такому підходу забезпечується постійне вдосконалення сервісів і підтримується їх конкурентоспроможність [6], [7].

Функціонування інструментів комунікації в реальному часі ґрунтується на їхній здатності підтримувати безперервний зв’язок між користувачами та тими, хто відповідає за вирішення їхніх запитів. Для забезпечення належної якості взаємодії застосовується спеціалізоване програмне забезпечення, яке координує передавання повідомлень, а також допоміжні модулі, що автоматизують частину рутинних дій [6], [7].

Обслуговування звернень потребує фахового підходу, оскільки оброблення великої кількості одночасних діалогів є складним процесом, що вимагає високої швидкості реагування та стійкої уваги. Успішне функціонування таких систем залежить від уміння оперативно виявляти проблему, пропонувати її рішення та підтримувати продуктивну комунікацію [6], [7].

Сучасне програмне середовище передбачає використання додаткових можливостей: прикріплення файлів, віддалену демонстрацію та перенаправлення запитів до відповідних спеціалістів. Такі функції сприяють гнучкості процесу та дозволяють розширювати сферу застосування систем оброблення повідомлень [6], [7].

Інтеграція інструментів комунікації безпосередньо в вебсторінки або мобільні додатки робить процес звернення максимально плавним: користувачу достатньо натиснути на відповідну панель у знайомому інтерфейсі. Відсутність необхідності переходити до сторонніх застосунків підвищує зручність і пришвидшує вирішення завдань [6], [7].

Правильна організація таких систем передбачає адаптацію зовнішнього вигляду елементів спілкування до стильових характеристик програмного продукту, а також забезпечення достатнього рівня підготовки персоналу, який взаємодіє з користувачами. Чіткі критерії часу відповіді та постійний моніторинг якості обслуговування формують позитивне враження від сервісу та мотивують звертатися повторно [6], [7].

Проактивне спрямування комунікації сприяє підвищенню зацікавленості: ініціалізація діалогу відбувається в момент, коли користувач виявляє певну активність або потребу в допомозі. Таке залучення часто усуває перешкоди та впливає на кінцевий результат взаємодії [6], [7].

Постійне спостереження за роботою системи комунікації в локальній мережі та її регулярне вдосконалення є необхідною умовою для підтримання належного рівня сервісу. Застосування засобів аналізу дозволяє визначати кількість контактів, оцінювати швидкість реакції, ефективність розв’язання звернень та рівень задоволення користувачів. Вивчення історії спілкування допомагає виявляти типові запити, повторювані труднощі та можливі напрями підвищення якості системи. Орієнтація на думку аудиторії й обґрунтовані висновки на основі даних забезпечують позитивні зміни та сприяють кращим результатам у функціонуванні таких сервісів [6], [7].

Рішення щодо способів обслуговування користувачів у цифровому середовищі можуть бути різними, і системи оброблення інформації для оперативного спілкування в локальній мережі мають чітко виражені переваги порівняно з іншими форматами. Моделі взаємодії через голосовий зв’язок вимагають очікування на з’єднання та не дозволяють обслуговувати багато запитів одночасно, тоді як текстове спілкування позбавлене цих обмежень і забезпечує доступ до архіву повідомлень для подальшого аналізу [6], [7].

Комунікація через електронну пошту часто розтягується в часі й не гарантує швидкого розв’язання проблем, у той час як обмін повідомленнями в режимі реального часу дає змогу працювати значно оперативніше й зручніше для користувачів, особливо коли йдеться про складні питання [6], [7].

Автоматизовані рішення на основі чат-ботів здатні впоратися з простими завданнями, однак вони не замінюють живого контакту, коли необхідні співпереживання, індивідуальний підхід та гнучке реагування. Саме тому людський фактор продовжує відігравати важливу роль у сервісних системах [6], [7].

Під час вибору платформи для впровадження подібних засобів враховується набір можливостей програмного забезпечення, його здатність до масштабування та адаптації до зростання навантаження. Важливо, щоб система гармонійно поєднувалася з наявною архітектурою вебсайту або клієнтського застосунку, не створювала технічних обмежень і забезпечувала безперервність користувацького досвіду. Крім функціональних критеріїв, увага приділяється фінансовим аспектам: різні постачальники пропонують моделі оплати за кількість користувачів або обсяг активних діалогів, що потребує ретельного планування витрат [6], [7].

Значення має й простота освоєння інтерфейсу, адже від швидкості адаптації операторів залежить результативність роботи всієї системи. Наявність навчальних матеріалів та підтримки дозволяє швидко досягти належного рівня сервісу. Зручність для кінцевого користувача та технічної команди стає вирішальним чинником успішного впровадження та подальшого розвитку системи опрацювання даних для спілкування в локальній мережі [6], [7].

Оцінка ефективності системи живого спілкування є важливою для розуміння того, наскільки вона виконує свої завдання та сприяє досягненню бізнес-цілей. Для цього застосовуються методи вимірювання продуктивності, які дозволяють визначити, наскільки швидко та якісно відбувається взаємодія користувачів із системою, а також наскільки задоволені її учасники. Використання аналітичних інструментів допомагає отримати чітке уявлення про роботу системи та виявити області, що потребують покращення [6], [7].

Основні показники ефективності відображають швидкість реагування системи та операторів на запити, обсяг оброблених комунікацій, тривалість розв’язання проблем і рівень задоволеності користувачів. Швидкість відповіді демонструє оперативність реагування, а кількість взаємодій дозволяє оцінити навантаження на команду. Час вирішення звернень відображає ефективність обробки запитів, тоді як оцінки задоволеності показують, наскільки користувачі отримують бажаний результат від взаємодії [6], [7].

Застосування аналітики дозволяє глибше зрозуміти діяльність операторів, виявити сильні сторони та визначити сфери, які потребують додаткової уваги або навчання. Завдяки таким даним роботу команди можна організувати більш збалансовано, розподіляючи запити таким чином, щоб підвищити загальну продуктивність і забезпечити стабільно високий рівень обслуговування. Впровадження аналітичного підходу сприяє не лише підвищенню ефективності агентів, але й покращенню досвіду користувачів, що взаємопов’язано з успіхом усієї системи живого спілкування [6], [7].

Оцінка користувацького досвіду через опитування та збір відгуків виступає важливим механізмом для визначення того, наскільки ефективно функціонує система живого спілкування з точки зору користувачів. Такі опитування можуть виконуватися безпосередньо під час сеансу чату або надсилатися після нього електронною поштою для отримання більш детальної інформації. Зібрані дані дозволяють оцінити рівень задоволеності, професіоналізм агентів та загальну якість взаємодії. Аналіз цих відгуків відкриває можливості для коригування процесів та підвищення якості обслуговування, сприяючи формуванню позитивного досвіду користувачів [6], [7].

Для систематизації та інтерпретації отриманих результатів активно використовуються спеціалізовані інструменти звітності та візуалізації даних. Вони забезпечують наочне відображення показників, дозволяючи швидко виявляти тенденції, порівнювати ефективність агентів і оцінювати загальний рівень задоволеності клієнтів. Використання таких засобів полегшує прийняття рішень на основі конкретних даних і дає змогу постійно оптимізувати роботу живого чату, підвищуючи його продуктивність та користь для бізнесу [6], [7].

Визначено, що система опрацювання даних для спілкування у локальній мережі забезпечує обмін повідомленнями між користувачами, під’єднаними до однієї мережі без використання зовнішніх серверів чи доступу до Інтернету. Така система обробляє, передає та зберігає дані у межах

внутрішньої мережевої інфраструктури, забезпечуючи швидку та захищену комунікацію між пристроями.

1.2 Аналіз систем опрацювання даних для спілкування у локальній мережі

Проаналізуємо системи опрацювання даних для спілкування у локальній мережі [8]-[14].

Mesh є програмною системою для захищеного обміну повідомленнями, яка функціонує на основі прямого взаємозв'язку між користувачами без проміжних серверів (рис. 1.1). Її реалізація орієнтована на повну конфіденційність, що забезпечується використанням криптографічних методів та механізмів приховування ідентичності через анонімні мережеві канали. Спілкування та передавання даних можуть здійснюватися як у звичайних мережевих середовищах з доступом до глобальної мережі, так і всередині приватних локальних сегментів, де зовнішнє підключення відсутнє [8], [12].

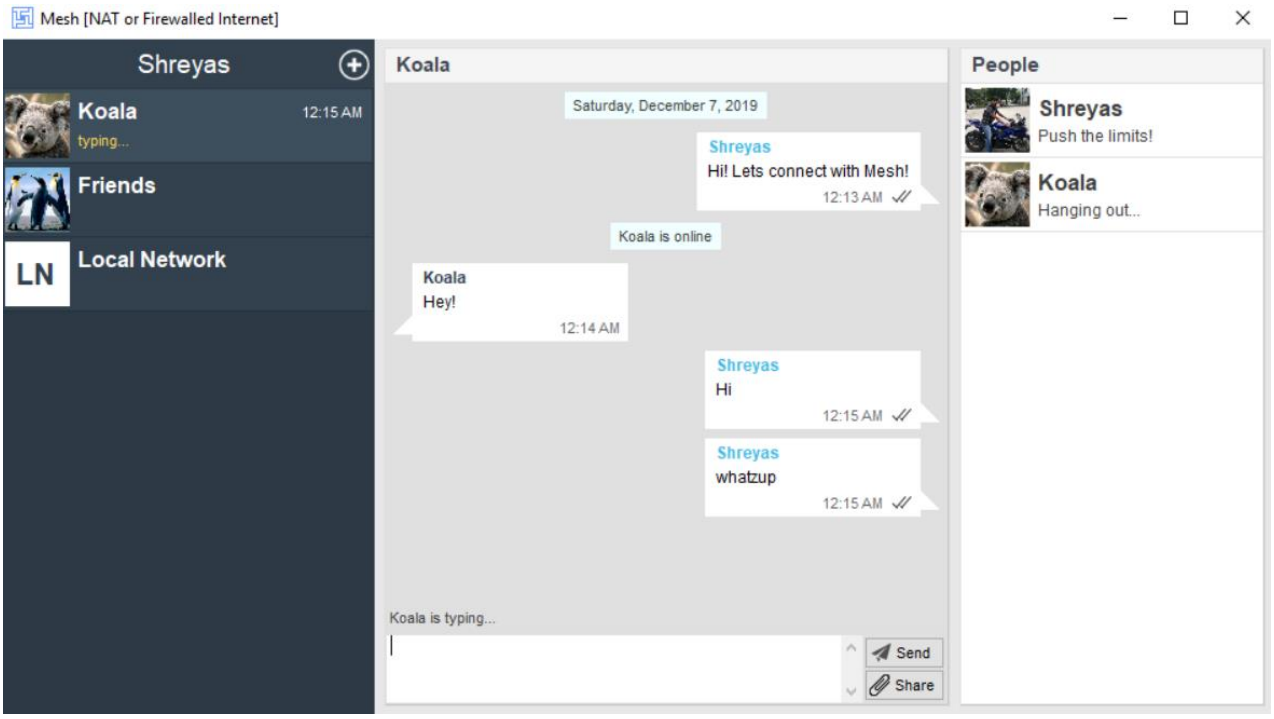

Рисунок 1.1 – Програмна система Mesh [12]

Система Mesh підтримує індивідуальні та групові діалоги, а також пересилання файлів без відкриття інформації стороннім структурам. Платформа розповсюджується у статусі відкритого програмного забезпечення відповідно до вільної ліцензії, а її код може бути вивчений та змінений будь-ким. Застосунок призначено для використання у середовищі настільних операційних систем та пропонує створення облікових записів, що не розкривають власника, завдяки застосуванню анонімних транспортних технологій. Захист профілів будується на основі асиметричної криптосхеми, тоді як обмін повідомленнями відбувається за допомогою стійкого симетричного шифрування з перевіркою достовірності. Обмін маршрутами здійснюється через розподілену структуру, яка дозволяє встановлювати з'єднання без центральної точки керування, а дані про активність не накопичуються та не ведуться, що виключає можливість відстеження взаємодії користувачів [8], [12].

Основними характеристиками програмної системи Mesh є такі [8], [12]:

– p2p-взаємодія: система Mesh працює без центрального сервера, забезпечуючи пряме з'єднання між користувачами та розподілене управління мережею [8], [12];

– підтримка локальної мережі: програмна система Mesh забезпечує можливість обміну повідомленнями та файлами навіть без доступу до глобальної мережі, що гарантує автономність системи [8], [12];

– наскрізне шифрування: усі дані шифруються на всіх етапах передавання, що забезпечує захист від перехоплення інформації [8], [12];

– анонімність користувачів: у програмній системі Mesh використовується технологія приховування реальних ідентифікацій з допомогою спеціальних мережевих механізмів, зокрема Tor [8], [12];

– криптографічний захист профілів: створення та захист облікових записів відбувається з використанням сучасних асиметричних ключів, що ускладнює підроблення чи злам профілю [8], [12];

– захищений протокол передавання: у програмній системі Mesh дані шифруються стійким алгоритмом, який гарантує їх конфіденційність та цілісність під час обміну [8], [12];

– відсутність зберігання метаданих: система не фіксує технічну інформацію щодо передачі повідомлень, що виключає можливість відстеження користувачів [8], [12];

– відкритий вихідний код: програмний продукт Mesh доступний для аналізу, модифікації та подальшого розвитку будь-якою зацікавленою спільнотою [8], [12];

– підтримка індивідуальних і групових чатів: забезпечується можливість особистої та колективної взаємодії між користувачами в межах однієї мережі [8], [12];

– безпечна передача файлів: система Mesh дає змогу обмінюватися файлами без ризику їх перехоплення або модифікації сторонніми [8], [12];

– робота на настільних ОС: програмне забезпечення Mesh функціонує в середовищі персональних комп’ютерів, що розширює можливості для використання у локальних корпоративних мережах [8], [12];

– розподілена хеш-таблиця: для маршрутизації використовується технологія DHT, яка дозволяє знаходити вузли та обмінюватися інформацією без централізованих ресурсів [8], [12].

Серед недоліків програмної системи Mesh можна відзначити обмеженість у виборі платформ для встановлення, оскільки застосунок доступний лише для настільних операційних систем, що унеможливлює використання на мобільних пристроях та суттєво звужує потенційне коло користувачів. Також певні складнощі можуть виникати у процесі початкового налаштування, особливо під час конфігурації анонімного з’єднання через спеціалізовані мережеві технології, що потребує додаткових знань і навичок. Робота без централізованого сервера призводить до залежності від стабільності самої мережі, тому погана якість локального з’єднання може негативно впливати на швидкість передавання даних. Крім цього, відсутність

широкої підтримки користувачів і повільніший розвиток у порівнянні з масовими комунікаційними платформами обмежують поширення системи, що, своєю чергою, позначається на доступності співрозмовників і загальній популярності рішення [8], [12].

Програмна система KouChat (рис. 1.2) являє собою просте у використанні рішення для організації локального обміну повідомленнями між користувачами всередині однієї мережі. Його реалізація орієнтована на швидкий запуск без складних параметрів, тому користувач може одразу розпочати спілкування після встановлення застосунку. Система KouChat працює на декількох популярних настільних і мобільних платформах, оскільки створена за допомогою універсальних технологічних засобів, що забезпечують її кросплатформність [8], [13].

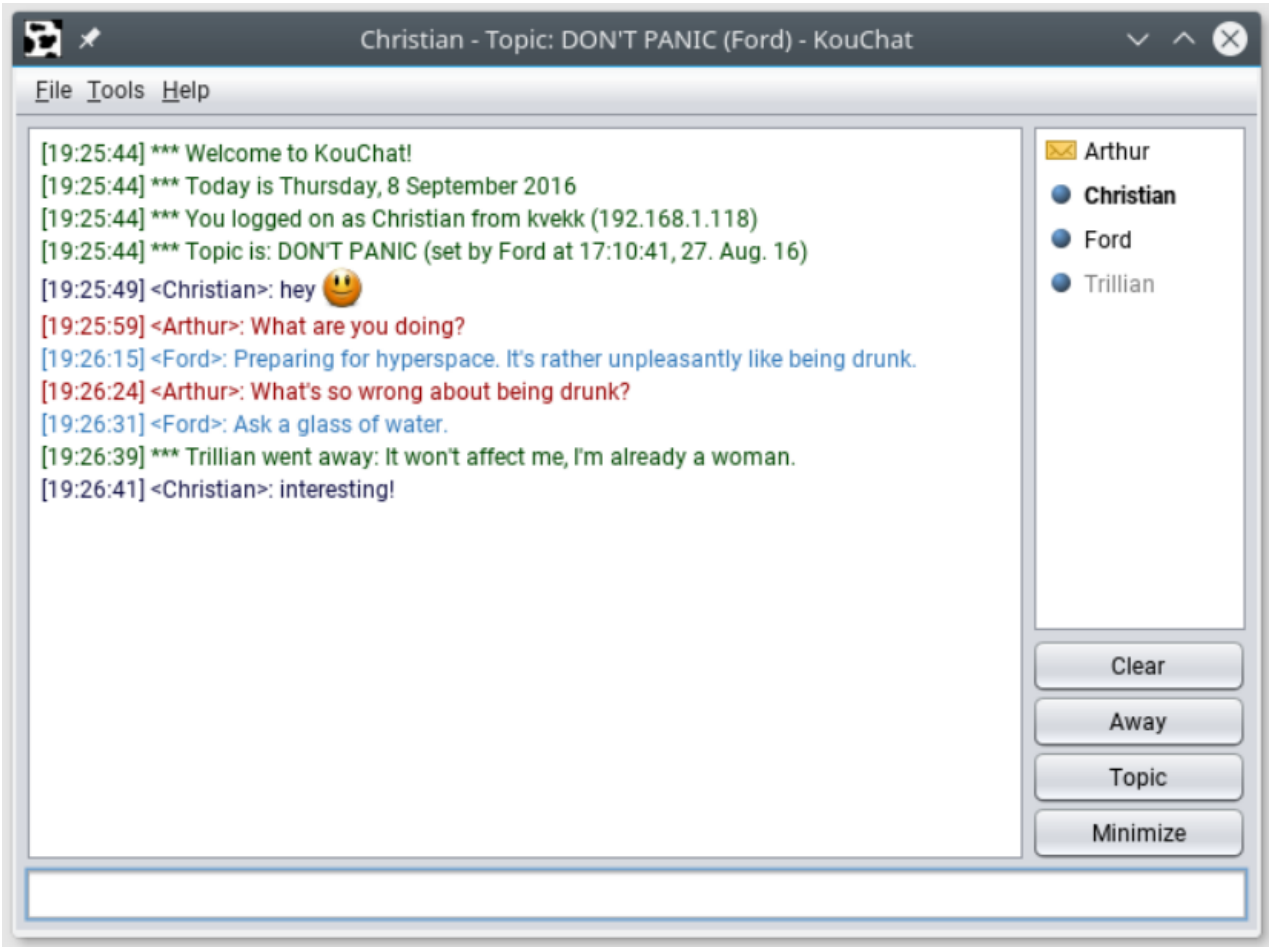

Рисунок 1.2 – Програмна система KouChat [13]

У програмі KouChat передбачено гнучке налаштування зовнішнього вигляду, можливість реагувати на отримані повідомлення за допомогою спливаючих оповіщень та швидкий доступ до основних функцій через панель операційної системи [8], [13].

Спілкування відбувається як у загальному просторі з одразу доступними учасниками, так і у форматі приватного діалогу між вибраними співрозмовниками. Програма надає можливість обирати індивідуальне ім’я відображення, змінювати стиль власних повідомлень, перевіряти активність інших користувачів під час введення тексту. Підтримується передавання файлів, що дає змогу швидко обмінюватися документами та мультимедійним вмістом. Передбачено також режим тимчасової неактивності, коли користувач не хоче отримувати повідомлення [8], [13].

Застосунок доступний для модифікації та розвитку спільнотою, оскільки поширюється як вільне програмне забезпечення з відкритими вихідними кодами. Текстова передача підтримує різні мовні символи, що дозволяє безперешкодно спілкуватися користувачам із різних мовних середовищ. Можливість ведення журналу діалогів допомагає зберігати історію обміну повідомленнями, а окремий консольний інтерфейс забезпечує альтернативний спосіб взаємодії з системою для досвідчених користувачів [8], [13].

Програмна система KouChat має такі особливості [8], [13]:

– кросплатформність: застосунок KouChat працює на різних операційних системах, що забезпечує можливість обміну повідомленнями між пристроями з різними програмними платформами [8], [13];

– простота використання: запуск і налаштування не потребують спеціальних знань, тому користувач може почати роботу одразу після встановлення [8], [13];

– групове спілкування: усі користувачі локальної мережі можуть взаємодіяти у спільному чаті KouChat, отримуючи повідомлення в режимі реального часу [8], [13];

– приватні діалоги: передбачена можливість індивідуальної комунікації між двома співрозмовниками без доступу сторонніх осіб [8], [13];

– використання псевдонімів: кожен користувач самостійно обирає ім’я, яке відображається іншим учасникам мережевої взаємодії [8], [13];

– підтримка оповіщень: система KouChat інформує про нові повідомлення користувачів, використовуючи функції області сповіщень та панелі завдань [8], [13];

– передавання файлів: користувачі KouChat можуть обмінюватися файлами без підключення до зовнішніх серверів або Інтернету [8], [13];

– налаштування інтерфейсу: є можливість змінювати тему та кольори оформлення для зручності і персоналізації роботи [8], [13];

– індикація активності набору тексту: у чаті KouChat відображається стан співрозмовника, коли він вводить повідомлення [8], [13];

– ведення історії: застосунок KouChat зберігає журнал листування, що дозволяє переглядати попередні повідомлення у будь-який момент [8], [13];

– режим відсутності: користувач може встановити статус, що обмежує отримання сповіщень та переривання під час роботи [8], [13];

– консольний режим: для досвідчених користувачів доступний альтернативний інтерфейс керування через командний контур [8], [13];

– відкритий вихідний код: поширення за вільною ліцензією дозволяє спільноті покращувати й розвивати застосунок [8], [13].

Серед недоліків програмної системи KouChat можна відзначити обмежену функціональність у порівнянні з сучасними месенджерами, відсутність підтримки передачі файлів у великих обсягах та обмежені можливості щодо шифрування даних, що знижує рівень безпеки під час обміну повідомленнями. Також додаток не має кросплатформної підтримки для мобільних операційних систем, що унеможливлює його використання на широкому колі пристроїв. Інтерфейс програми виглядає застарілим і може бути незручним для користувачів, які звикли до сучасних UX/UI рішень. Крім того, відсутність централізованого серверного компонента хоч і забезпечує

повну незалежність від мережі Інтернет, однак водночас обмежує можливості для синхронізації даних та реалізації додаткових сервісів [8], [13].

Програмна система Khernet (рис. 1.3) являє собою засіб для локального обміну повідомленнями, орієнтований на простоту використання та автономність роботи в межах внутрішньої мережі. Програма поширюється на умовах відкритої ліцензії, що дає змогу вільно застосовувати її для особистих та професійних цілей, змінювати вихідний код і передавати іншим користувачам, водночас всі ризики щодо її застосування покладаються на користувача [8], [14].

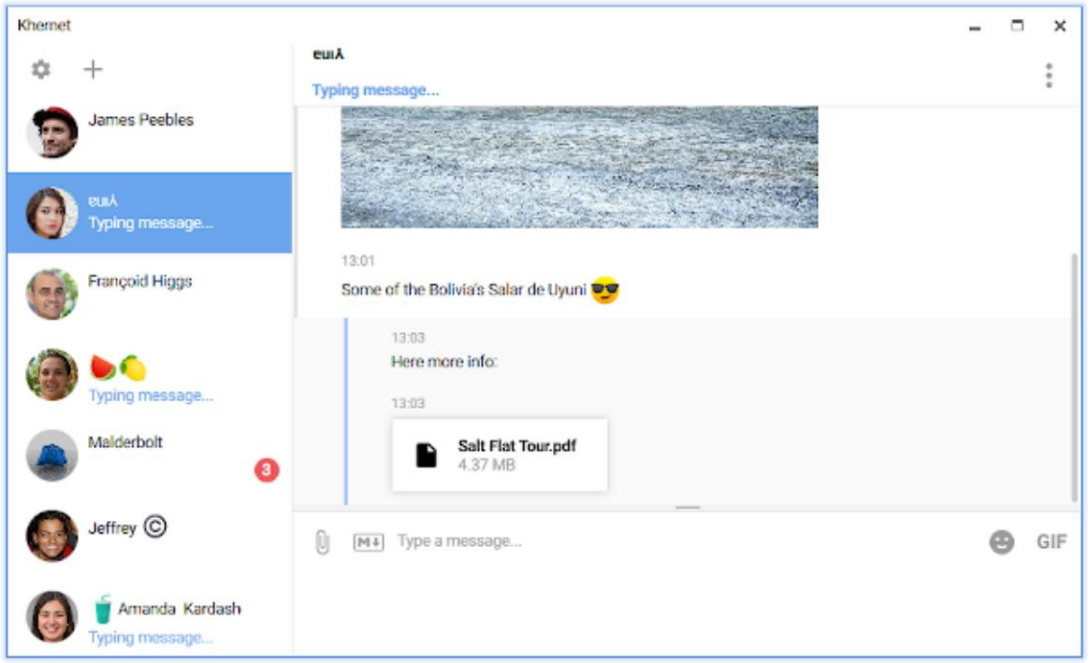

Рисунок 1.3 – Програмна система Khernet [14]

Застосунок Khernet не вимагає серверної інфраструктури, функціонує як окремий клієнт і здатний запускатися без інсталяції, що робить його зручним у середовищах із обмеженим налаштуванням обладнання. Система підтримує захист даних шляхом наскрізного шифрування, дозволяє передавати текстові повідомлення із використанням форматування та графічних елементів, а також обмінюватися мультимедійними й іншими файлами. За потреби можливе використання як портативного варіанта, так і

режиму встановлення, а оновлення можуть виконуватися як у мережі, так і автономно [8], [14].

Програмна система Khernet має такі особливості [8], [14]:

– безкоштовне програмне забезпечення: застосунок Khernet може використовуватися без потреби оплати та доступний для широкого кола користувачів [8], [14];

– відкритий вихідний код: програмна система Khernet надає можливість змінювати програму, аналізувати її роботу та розповсюджувати модифіковані версії [8], [14];

– робота без встановлення: портативний режим програмної системи Khernet дає змогу запускати застосунок одразу після завантаження без інсталяції в систему [8], [14];

– автономність від серверів: функціонування програмної системи Khernet забезпечується у межах локальної мережі без використання зовнішньої серверної інфраструктури [8], [14];

– захищеність комунікацій: дані у програмній системі Khernet передаються з використанням наскрізного шифрування, що підвищує рівень конфіденційності [8], [14];

– підтримка різних типів повідомлень: надсилання тексту, зображень, GIF, відео та інших файлів доступне під час спілкування [8], [14];

– зручне позначення контактів: користувачі програмної системи Khernet можуть використовувати емодзі у своїх іменах для швидкої ідентифікації співрозмовників [8], [14];

– варіативність оновлень: програмна система Khernet підтримує можливість встановлення оновлень як онлайн, так і офлайн забезпечує стабільне використання у будь-яких умовах [8], [14].

Серед недоліків програмної системи Khernet можна відзначити обмеженість функціональності порівняно з повноцінними корпоративними рішеннями, відсутність підтримки хмарної синхронізації та розширених інструментів адміністрування, а також залежність від локальної мережі, що

унеможливлює спілкування поза її межами. Крім того, інтерфейс та загальна архітектура можуть потребувати доопрацювання для підвищення зручності роботи користувачів та масштабованості ресурсу в умовах великої кількості учасників [8], [14].

Порівняльну характеристику систем опрацювання даних для спілкування у локальній мережі наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння систем опрацювання даних для спілкування у локальній мережі

Критерій порівняння	Mesh [12]	KouChat [13]	Khernet [14]
Простота інсталяції та запуску	+	+–	+
Портативність	+–	+–	+
Шифрування повідомлень	+	–	+
Підтримка кількох платформ	+	+	+–
Запис історії чату	+	+	+–
Можливість змінювати тему/інтерфейс	–	+	–

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить багато систем опрацювання даних для спілкування у локальній мережі. Проте деякі системи опрацювання даних для спілкування у локальній мережі є досить дорогими, крім того їх використання пов’язано з потребою в постійному технічному обслуговуванні мережевої інфраструктури та серверного обладнання. Відмова будь-якого елемента або збій в енергопостачанні можуть призвести до повної зупинки комунікації. Забезпечення належного рівня безпеки також вимагає професійного адміністрування, оскільки локальні системи, без оновлень і моніторингу, можуть стати вразливими до внутрішніх загроз. Крім того, масштабування такої системи опрацювання даних при зростанні кількості користувачів або

географічному розширенні організації може потребувати значних матеріальних і часових витрат. Тому актуальною є розробка системи опрацювання даних для спілкування у локальній мережі.

При розробці системи опрацювання даних для спілкування у локальній мережі необхідно забезпечити такі функціональні вимоги:

– підтримка можливості авторизації користувачів;

– підтримка можливості відображення переліку користувачів системи для спілкування;

– можливість відображення статусу користувачів (доступний до спілкування, недоступний для спілкування, офлайн);

– можливість спілкування з обраними користувачами за допомогою текстових повідомлень.

1.3 Висновки за розділом 1

Визначено, що система опрацювання даних для спілкування у локальній мережі забезпечує обмін повідомленнями між користувачами, під’єднаними до однієї мережі без використання зовнішніх серверів чи доступу до Інтернету. Така система обробляє, передає та зберігає дані у межах внутрішньої мережевої інфраструктури, забезпечуючи швидку та захищену комунікацію між пристроями.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато систем опрацювання даних для спілкування у локальній мережі. Проте деякі системи опрацювання даних для спілкування у локальній мережі є досить дорогими, крім того їх використання пов’язано з потребою в постійному технічному обслуговуванні мережевої інфраструктури та серверного обладнання. Відмова будь-якого елемента або збій в енергопостачанні можуть призвести до повної зупинки комунікації. Забезпечення належного рівня безпеки також вимагає професійного адміністрування, оскільки локальні системи, без оновлень і моніторингу,

можуть стати вразливими до внутрішніх загроз. Крім того, масштабування такої системи опрацювання даних при зростанні кількості користувачів або географічному розширенні організації може потребувати значних матеріальних і часових витрат. Тому актуальною є розробка системи опрацювання даних для спілкування у локальній мережі.

Сформульовано функціональні вимоги до системи опрацювання даних для спілкування у локальній мережі.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

На вибір мови програмування та інших засобів розробки суттєво впливає середовище функціонування програмної системи. Тому перед вибором мови та середовища розробки було проаналізовано операційні системи, у яких може функціонувати система опрацювання даних для спілкування у локальній мережі.

Визначено, для функціонування системи опрацювання даних для спілкування у локальній мережі доцільним є використання операційної системи Android [15], [16].

Вибір операційної системи Android для розробки та функціонування системи опрацювання даних для спілкування у локальній мережі є доцільним з кількох важливих причин [15], [16].

Передусім, Android володіє широкою доступністю та популярністю: більшість сучасних користувачів мають мобільні пристрої саме на цій платформі, що забезпечує легке впровадження системи без потреби у додатковому придбанні обладнання. Такий підхід сприяє зменшенню витрат та підвищенню зручності використання. Операційна система Android підтримує великий спектр апаратних різновидів, від смартфонів до планшетів, що дозволяє адаптувати функціональність під різні сценарії роботи в організації [15], [16].

Android надає розвинені засоби розробки, включно з офіційним середовищем Android Studio та широкою екосистемою бібліотек, інструментів і документації. Це дає можливість швидко створювати, тестувати й удосконалювати застосунки з урахуванням вимог до локального обміну даними. Крім того, система підтримує сучасні мережеві технології, зокрема Wi-Fi Direct, Bluetooth та локальні TCP/IP-з’єднання, що дає змогу організувати прямий обмін повідомленнями між пристроями без доступу до Інтернету [15], [16].

Окрему перевагу становить відкритість Android як платформи. Вона дозволяє інтегрувати програмне забезпечення з іншими інструментами організації, налаштовувати мережеві параметри та контролювати політики безпеки відповідно до специфічних потреб. Система також вирізняється гнучкістю щодо розширення функціональності, включно з використанням мультимедійних можливостей, внутрішніх датчиків пристрою та механізмів шифрування даних [15], [16].

Обґрунтування вибору операційної системи для функціонування системи опрацювання даних для спілкування у локальній мережі наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору операційної системи для функціонування системи опрацювання даних для спілкування у локальній мережі

Критерій порівняння операційних систем	Операційні системи		
	Android	Windows	iOS
Поширеність мобільних пристроїв серед користувачів	+	–	+
Вартість розгортання на мобільних пристроях	+	+–	+–
Простота оновлення та масштабування рішення	+	+–	+–
Розвинені інструменти для швидкої розробки застосунків	+	+	+–
Гнучкість у виборі способів інсталяції застосунку	+	+–	–

Отже, для функціонування системи опрацювання даних для спілкування у локальній мережі обрано операційну систему Android, яка

забезпечує оптимальне поєднання доступності, функціональної розвиненості, мережевої сумісності й економічної ефективності, що робить його раціональним вибором для реалізації системи опрацювання даних для спілкування у локальній мережі [15], [16].

При розробці системи опрацювання даних для спілкування у локальній мережі було використано мову програмування Kotlin [17], [18].

Вибір мови програмування Kotlin для створення системи опрацювання даних для спілкування у локальній мережі є обґрунтованим як з технічної, так і з практичної точки зору [17], [18].

Передусім, мова програмування Kotlin є офіційною мовою розробки під Android, що забезпечує найкращу сумісність із функціональними можливостями операційної системи та доступ до всіх нативних API. Завдяки цьому створення мережевих сервісів, фонових процесів та інтеграція з апаратними ресурсами пристроїв Android відбувається ефективно й без додаткових накладних витрат [17], [18].

Важливою перевагою Kotlin є його сучасна й безпечна архітектура. Мова мінімізує типові помилки під час роботи з пам’яттю та об’єктами, зокрема проблему NullPointerException, завдяки системі null-безпеки. Це особливо суттєво для програм, що працюють у реальному часі та потребують стабільності під час обміну даними в мережі [17], [18].

Kotlin підтримує асинхронне програмування, що дозволяє ефективно обробляти мережеві запити, не блокуючи роботу застосунку. Такий підхід покращує продуктивність системи комунікації й забезпечує швидку реакцію навіть за значного навантаження [17], [18].

Екосистема Kotlin включає велику кількість готових бібліотек для роботи з мережевими протоколами, шифруванням, локальними базами даних, серіалізацією повідомлень. Це скорочує час розробки та забезпечує високу якість реалізації необхідної функціональності [17], [18].

Крім того, Kotlin повністю сумісний із Java, що дозволяє без проблем використовувати перевірені інструменти та бібліотеки й водночас створювати

більш компактний і зрозумілий код. Це спрощує підтримку та розширення застосунку в майбутньому [17], [18].

Обґрунтування вибору мови програмування для розробки системи опрацювання даних для спілкування у локальній мережі наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору мови програмування для розробки системи опрацювання даних для спілкування у локальній мережі

Критерій порівняння мов програмування	Мова програмування		
	Kotlin	Java	C#
Нативна підтримка Android	+	+–	–
Підтримка асинхронності та багатопоточності	+	+–	+
Безпека типів	+	+–	+–
Продуктивність при роботі з мобільними мережевими API	+	+	+–
Можливість швидкої розробки та тестування	+	+–	+–

Отже, для реалізації системи опрацювання даних для спілкування у локальній мережі обрано мову програмування Kotlin, яка поєднує безпеку, високу продуктивність, зручність розробки і нативну підтримку Android, що робить цю мову оптимальним вибором для створення відповідної програми. Важливим є те, що Kotlin є офіційною мовою розробки під Android, що забезпечує найкращу сумісність із функціональними можливостями операційної системи та доступ до всіх нативних API. Завдяки цьому створення мережевих сервісів, фонових процесів та інтеграція з апаратними ресурсами пристроїв Android відбувається ефективно й без додаткових накладних витрат.

2.2 Вибір середовища розробки для створення системи опрацювання даних для спілкування у локальній мережі

В якості середовища розробки для створення системи опрацювання даних для спілкування у локальній мережі було обрано середовище Android Studio [19], [20].

Середовище розробки Android Studio є офіційним середовищем, створеним і підтримуваним Google, що гарантує повну сумісність із платформою та доступ до всіх актуальних інструментів, API і сервісів. Це дозволяє розробникам максимально ефективно використовувати можливості мобільних пристроїв для організації локальної мережевої взаємодії [19], [20].

Однією з ключових переваг Android Studio є наявність інструментів для створення, тестування та налагодження мережевих застосунків. Емулятори забезпечують моделювання різних типів з'єднань (Wi-Fi, Bluetooth, локальні протоколи), що сприяє точній перевірці функціональності системи ще на етапі розробки. Вбудовані профайлери дозволяють контролювати використання ресурсів, що особливо важливо для застосунків, які повинні працювати стабільно в режимі реального часу [19], [20].

Android Studio підтримує сучасні технології розробки: мову Kotlin, Jetpack-бібліотеки, архітектурні патерни (MVVM, Clean Architecture), механізми асинхронної обробки даних. Завдяки цьому реалізація локального обміну повідомленнями, шифрування, збереження та синхронізація даних відбувається швидше та якісніше [19], [20].

Крім того, середовище Android Studio надає зручні засоби UI-дизайну для створення інтерфейсів, адаптивних до різних форм-факторів пристроїв, що підвищує доступність системи для всіх користувачів локальної мережі. Інтегрована система контролю версій, автоматизація збирання (Gradle) та підтримка модульності дозволяють легко масштабувати проєкт і підтримувати його життєвий цикл [19], [20].

Android Studio має широку спільноту та великий обсяг навчальних ресурсів, що спрощує впровадження рішень і вирішення технічних проблем у процесі розробки [19], [20].

Обґрунтування вибору середовища розробки для створення системи опрацювання даних для спілкування у локальній мережі наведено у таблиці 2.3.

Таблиця 2.3 – Обґрунтування вибору середовища розробки для створення системи опрацювання даних для спілкування у локальній мережі

Критерій порівняння середовищ розробки	Середовища розробки		
	Android Studio	IntelliJ IDEA	Eclipse
Офіційна підтримка Android та Kotlin	+	+–	–
Вбудовані емулятори та мережеве тестування	+	+–	–
Можливість комплексного налагодження застосунків	+	+	+–
Інструменти аналізу продуктивності та ресурсів	+	+–	+–
Простота налаштування середовища	+	+–	+–

Таким чином, для створення системи опрацювання даних для спілкування у локальній мережі обрано середовище Android Studio, яке є офіційним середовищем, створеним і підтримуваним Google, що гарантує повну сумісність із платформою та доступ до всіх актуальних інструментів, API і сервісів. Це дозволяє розробникам максимально ефективно використовувати можливості мобільних пристроїв для організації локальної

мережевої взаємодії. Крім того, Android Studio надає інструменти для створення, тестування та налагодження мережевих застосунків, а також має широку спільноту та великий обсяг навчальних ресурсів, що спрощує впровадження рішень і вирішення технічних проблем у процесі розробки.

2.3 Вибір засобів зберігання та опрацювання даних

Для зберігання та опрацювання даних було обрано систему управління базами даних Room [21], [22].

Система управління базами даних Room є ключовим елементом у створенні надійної та оптимізованої системи зберігання даних на мобільних пристроях, що працюють під управлінням Android. Система забезпечує прозоре перетворення об’єктів Kotlin у записи бази даних і навпаки, що значно спрощує взаємодію з даними та робить код більш читабельним і підтримуваним. Завдяки цьому зменшується ймовірність логічних помилок при маніпуляціях із записами та підвищується загальна стійкість застосунку до некоректних даних [21], [22].

Room дозволяє ефективно реалізовувати складні запити та відносини між таблицями без потреби занурюватися у деталі низькорівневого SQL-коду, що скорочує час розробки і тестування. Використання Room полегшує роботу з транзакціями, забезпечуючи атомарність операцій і підтримку цілісності даних, навіть при одночасному доступі кількох потоків або користувачів. Крім того, система зручно інтегрується з сучасними архітектурними патернами Android, забезпечуючи автоматичне сповіщення компонентів інтерфейсу про зміни в базі даних без додаткового коду [21], [22].

Завдяки підтримці реактивних бібліотек Room дозволяє будувати асинхронні потоки даних, що підвищує продуктивність і забезпечує плавну роботу програми навіть при інтенсивних операціях з базою. Використання Room сприяє стандартизації процесу роботи з даними в проєкті та створює умови для легкого масштабування і подальшого розвитку системи, що робить

її надійним і довгостроковим рішенням для будь-яких мобільних застосунків, що працюють із локальними даними [21], [22].

Вибір системи управління базами даних Room для створення системи опрацювання даних для спілкування у локальній мережі є доцільним через її тісну інтеграцію з платформою Android та ефективну роботу з локальними даними на мобільних пристроях. Room забезпечує високий рівень абстракції над SQLite, що дозволяє працювати з базою даних без необхідності написання великої кількості низькорівневого коду, водночас зберігаючи повний контроль над структурою даних та запитами. Це особливо важливо для систем, які потребують швидкого зберігання та обробки повідомлень, файлів або станів користувачів у режимі реального часу [21], [22].

Система автоматично перевіряє правильність SQL-запитів під час компіляції, що зменшує кількість потенційних помилок і підвищує стабільність застосунку. Room підтримує асинхронну роботу з даними за допомогою корутин Kotlin або RxJava, що дозволяє уникати блокування основного потоку програми і забезпечує плавну роботу інтерфейсу під час виконання операцій з базою даних. Крім того, Room пропонує зручні механізми для міграції схем та збереження цілісності даних при оновленні програми, що робить її придатною для довгострокового використання і масштабування [21], [22].

Використання системи управління базами даних Room також сприяє оптимальній інтеграції з архітектурними компонентами Android, такими як ViewModel та LiveData, що забезпечує автоматичне оновлення інтерфейсу при зміні даних. Завдяки цим властивостям система опрацювання даних набуває високої продуктивності, надійності та легкості в підтримці, що робить Room раціональним і ефективним вибором для реалізації локальної мережевої комунікації [21], [22].

Обґрунтування вибору засобів зберігання та опрацювання даних наведено у таблиці 2.4.

Таблиця 2.4 – Обґрунтування вибору засобів зберігання та опрацювання даних

Критерій порівняння систем управління базами даних	Системи управління базами даних		
	Room	SQLite	Realm
Простота роботи з об’єктами даних	+	–	+
Контроль схем і міграцій	+	+–	+–
Стабільність та надійність для локальних даних	+	+	+–
Продуктивність при роботі з великими обсягами локальних даних	+	+–	+
Простота інтеграції з архітектурними патернами Android	+	–	+–

Таким чином, для зберігання та опрацювання даних обрано систему управління базами даних Room, яка поєднує повну інтеграцію з Android, підтримку Kotlin і сучасних архітектурних компонентів, високу продуктивність при роботі з локальними даними, асинхронність та надійність, а також зручні механізми управління схемами та міграціями. У результаті використання Room забезпечує легке створення, масштабування та підтримку стабільної системи опрацювання даних для локальної комунікації.

2.4 Висновки за розділом 2

Для функціонування системи опрацювання даних для спілкування у локальній мережі обрано операційну систему Android, яка забезпечує оптимальне поєднання доступності, функціональної розвиненості, мережевої

сумісності й економічної ефективності, що робить його раціональним вибором для реалізації системи опрацювання даних для спілкування у локальній мережі.

Для реалізації системи опрацювання даних для спілкування у локальній мережі обрано мову програмування Kotlin, яка поєднує безпеку, високу продуктивність, зручність розробки і нативну підтримку Android, що робить цю мову оптимальним вибором для створення відповідної програми.

Для створення системи опрацювання даних для спілкування у локальній мережі обрано середовище Android Studio, яке є офіційним середовищем, створеним і підтримуваним Google, що гарантує повну сумісність із платформою та доступ до всіх актуальних інструментів, API і сервісів. Це дозволяє розробникам максимально ефективно використовувати можливості мобільних пристроїв для організації локальної мережевої взаємодії. Крім того, Android Studio надає інструменти для створення, тестування та налагодження мережевих застосунків, а також має широку спільноту та великий обсяг навчальних ресурсів, що спрощує впровадження рішень і вирішення технічних проблем у процесі розробки.

Для зберігання та опрацювання даних обрано систему управління базами даних Room, яка поєднує повну інтеграцію з Android, підтримку Kotlin і сучасних архітектурних компонентів, високу продуктивність при роботі з локальними даними, асинхронність та надійність, а також зручні механізми управління схемами та міграціями. У результаті використання Room забезпечує легке створення, масштабування та підтримку стабільної системи опрацювання даних для локальної комунікації.

3 ОПИС СИСТЕМИ ОПРАЦЮВАННЯ ДАНИХ

3.1 Структура системи опрацювання даних для спілкування у локальній мережі

Структуру системи опрацювання даних для спілкування у локальній мережі наведено на рис. 3.1.

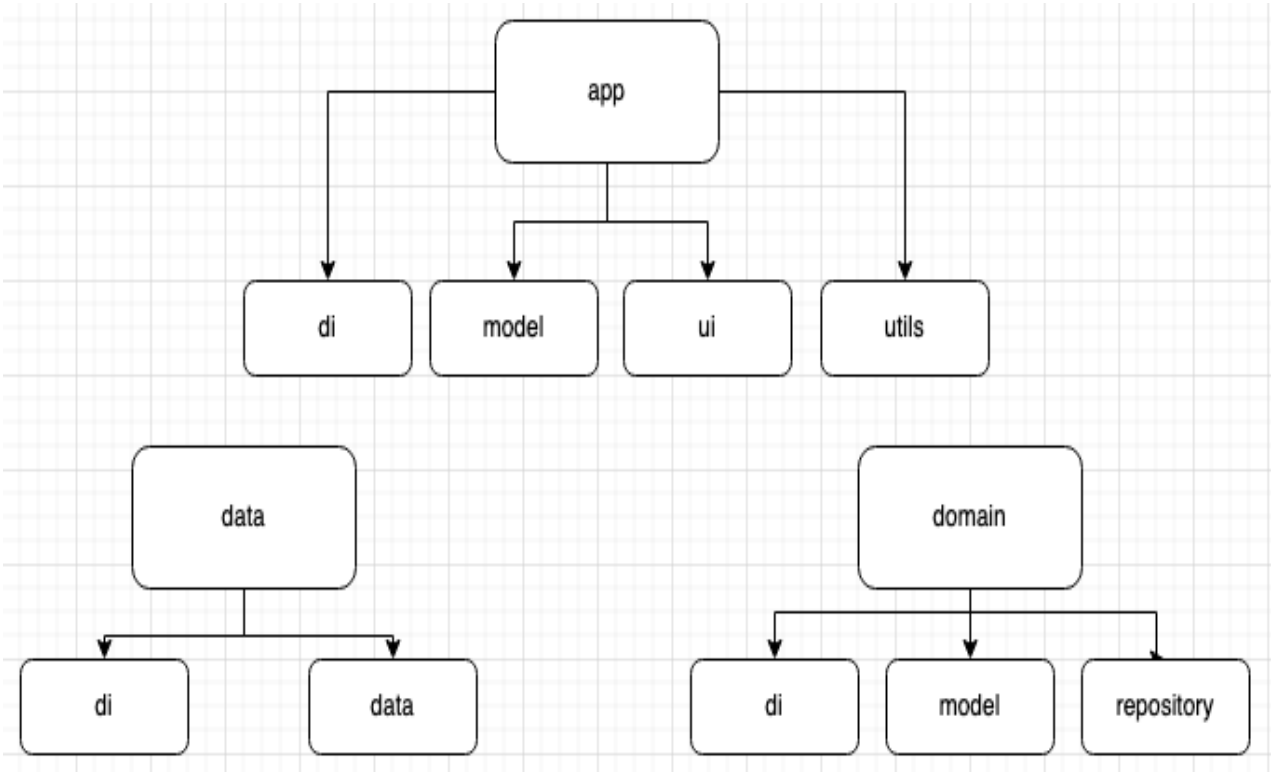

Рисунок 3.1 – Структура системи опрацювання даних для спілкування у локальній мережі

Як видно з рис. 3.1, структура системи опрацювання даних для спілкування у локальній мережі передбачає логічне розділення її складових на три пакети: пакет app для взаємодії з користувачем та представленням інформації, що відображається на екрані мобільного пристрою; пакет data відповідає за роботу з даними та поєднує в собі реалізацію механізмів збереження та отримання інформації; пакет domain виконує функцію логічного посередника між джерелами даних і компонентами, які їх використовують. Кожен пакет виконує власну роль у забезпеченні коректного

функціонування застосунку. Така організація сприяє підвищенню зручності розробки, підтримуваності та масштабованості системи, а також полегшує поділ відповідальності між програмними компонентами.

Пакет app присвячений взаємодії з користувачем та представленням інформації, що відображається на екрані мобільного пристрою. У ньому зосереджені модулі, що відповідають за візуальні елементи, а також класи, які забезпечують життєвий цикл застосунку і доступ до необхідних зовнішніх залежностей. Тут же зберігаються допоміжні компоненти, створення яких відбувається одноразово та надалі використовується протягом усієї роботи застосунку.

Пакет data відповідає за роботу з даними та поєднує в собі реалізацію механізмів збереження та отримання інформації, яка циркулює між локальною базою даних і серверними джерелами. Він забезпечує надходження та передачу даних у необхідному форматі, а також бере на себе налаштування залежностей для тих частин системи, що потребують взаємодії з інфраструктурою зберігання інформації.

Пакет domain виконує функцію логічного посередника між джерелами даних і компонентами, які їх використовують. Він містить абстрактні моделі, що описують структуру інформації у застосунку, а також реалізує механізми, котрі координують доступ до локальних ресурсів та до ресурсів, що знаходяться поза межами пристрою. Його призначення полягає в забезпеченні правильного узгодження взаємодії між базою даних, мережею та користувацькою частиною застосунку.

У підсумку така структурна модель системи опрацювання даних для спілкування у локальній мережі створює чітке розмежування відповідальностей: інтерфейс користувача, робота з даними та логіка їх обробки функціонують незалежно, але водночас узгоджено, що підвищує ефективність реалізації й подальшого супроводу програмного продукту.

3.2 Функціонування системи опрацювання даних для спілкування у локальній мережі

Функціонування системи опрацювання даних для спілкування у локальній мережі подамо за допомогою схеми, зображеної на рис. 3.2.

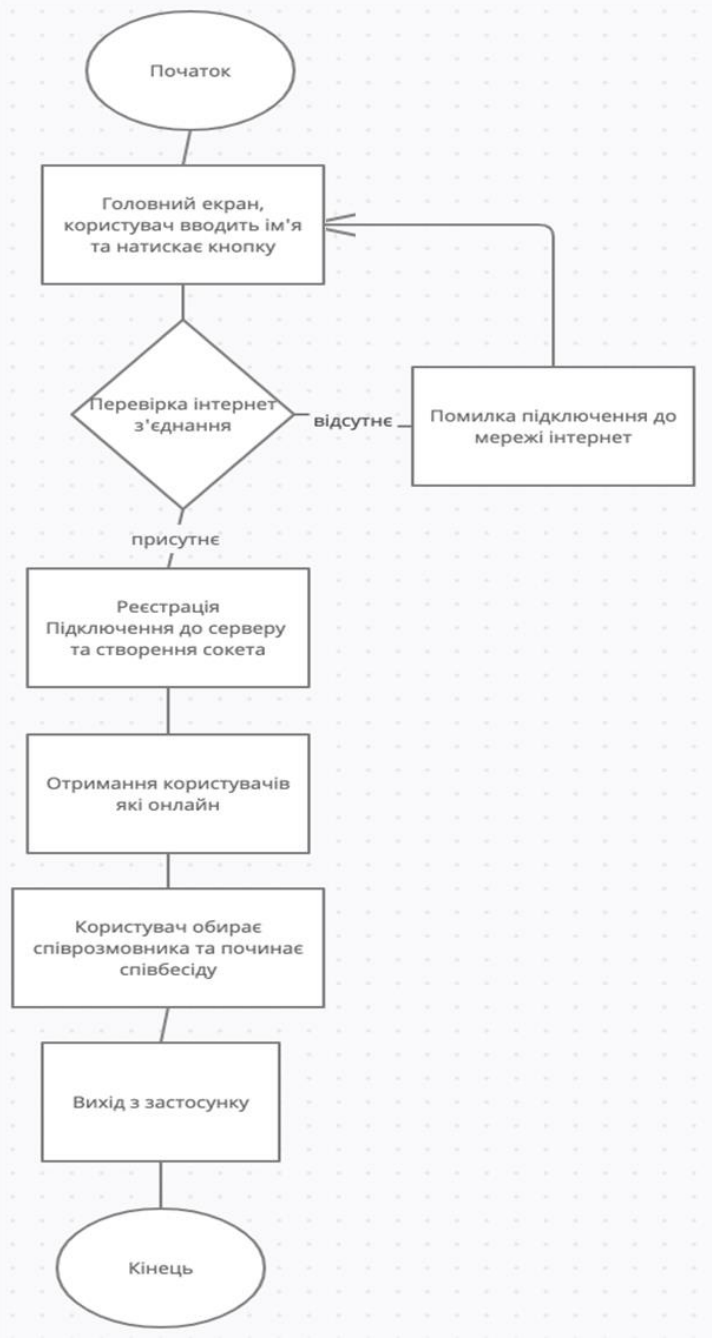

Рисунок 3.2 – Схема функціонування системи опрацювання даних для спілкування у локальній мережі

Функціонування системи опрацювання даних для спілкування у локальній мережі побудоване таким чином, щоб забезпечити максимально просту й зрозумілу взаємодію користувача із застосунком. Робота програми розпочинається з її запуску на мобільному пристрої, після чого користувач отримує можливість увійти до середовища обміну повідомленнями, ввівши будь-яке бажане ім’я. Цей етап ініціює процес реєстрації, у ході якого перевіряється доступність мережевого підключення: за умови наявності інтернету система встановлює зв’язок з серверною частиною та відкриває сокет для подальшого обміну даними. Якщо ж мережеве з’єднання відсутнє, користувача попереджають за допомогою діалогового повідомлення з проханням перевірити доступ до мережі.

Після успішного завершення початкового підключення інтерфейс системи опрацювання даних для спілкування у локальній мережі відображає список усіх активних користувачів, що перебувають онлайн у межах локальної мережі. Таким чином, кожен учасник має можливість обрати співрозмовника із запропонованого переліку й розпочати обмін повідомленнями у режимі реального часу. Передавання даних відбувається миттєво, адже система застосовує механізм прямої комунікації через сокетне з’єднання, що забезпечує високу швидкість взаємодії та мінімальні затримки обміну інформацією.

Після завершення спілкування користувач може припинити роботу із застосунком, просто закривши його. Усі процеси при цьому зупиняються, а з’єднання з сервером коректно розривається, що гарантує стабільність функціонування системи та відсутність зайвого навантаження на мережеві ресурси.

Загалом робота системи опрацювання даних для спілкування у локальній мережі характеризується логічною послідовністю кроків, які не потребують від користувача спеціальних навичок чи додаткових пояснень. Система опрацювання даних для спілкування у локальній мережі забезпечує безпечне, надійне та інтуїтивне середовище для спілкування між учасниками

локальної мережі, дозволяючи їм швидко встановлювати контакт і вести діалог у зручний спосіб.

3.3 Розробка бази даних системи опрацювання даних для спілкування у локальній мережі

У системі опрацювання даних для спілкування у локальній мережі основою зберігання інформації є локальна база даних, реалізована за допомогою СУБД Room. Такий підхід дозволяє створити структуроване, надійне та ефективне сховище, тісно пов’язане з архітектурними компонентами Android, що забезпечує високий рівень продуктивності та керованості даними. Room застосовує об’єктно-реляційну модель, що дає змогу описувати таблиці як об’єкти й автоматично опрацьовувати зв’язки між ними, зменшуючи ризик помилок при взаємодії з базою даних.

Схему бази даних системи опрацювання даних для спілкування у локальній мережі наведено на рис. 3.3.

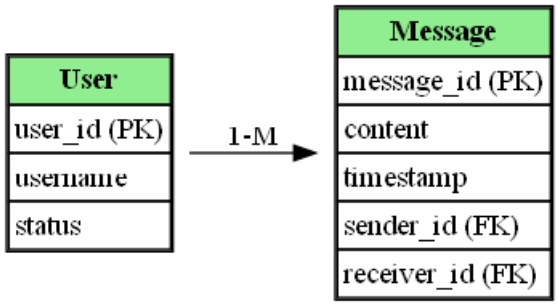

Рисунок 3.3 – Схема бази даних системи опрацювання даних для спілкування у локальній мережі

Логічна модель бази ґрунтується на фіксації користувачів локальної мережі та збереженні повідомлень, які пересилаються між ними. У структурі бази даних виділяються сутності, що відповідають за зберігання інформації про кожного користувача: його унікального ідентифікатора, відображуваного імені та стану активності у системі. Кожен запис у цій таблиці представляє

окремого учасника комунікаційного процесу й може бути оновлений у режимі реального часу відповідно до його підключення чи виходу з мережі.

Ще одна сутність описує повідомлення, які формуються та передаються в ході діалогу. Кожне повідомлення пов’язується з відправником і отримувачем за допомогою зовнішніх ключів, що дозволяє чітко простежувати логіку спілкування між конкретними користувачами. Це дає змогу відображати історію листування в належному порядку, забезпечувати збереження контексту бесіди та при необхідності повертатися до попередньо переданої інформації.

Зв’язок між таблицями має вигляд відношення «один до багатьох», де окремий користувач може мати великий перелік повідомлень, що були ним відправлені або отримані. Завдяки такому підходу забезпечується повна узгодженість структури бази з логічною організацією процесу комунікації. Room автоматично керує оновленням і видаленням пов’язаних записів, що підтримує цілісність і стабільність інформації.

У межах цієї системи локальна база даних відіграє ключову роль у відображенні стану мережевої взаємодії та збереженні інформації навіть у випадках, коли підключення до сервера тимчасово відсутнє. Її логічна структура формує фундамент для швидкого пошуку, відображення й синхронізації даних, що забезпечує безперервність та якість процесу спілкування у локальній мережі.

3.4 Проєктування інтерфейсу системи опрацювання даних для спілкування у локальній мережі

Проєктування інтерфейсу взаємодії користувача з системою опрацювання даних для спілкування у локальній мережі виконано з урахуванням вимог технічного завдання. При розробленні системи опрацювання даних для спілкування у локальній мережі враховані функціональні вимоги до програми:

– підтримка можливості авторизації користувачів;

– підтримка можливості відображення переліку користувачів системи для спілкування;

– можливість відображення статусу користувачів (доступний до спілкування, недоступний для спілкування, офлайн);

– можливість спілкування з обраними користувачами за допомогою текстових повідомлень.

Інтерфейс головного екрану системи опрацювання даних для спілкування у локальній мережі наведено на рис. 3.4.

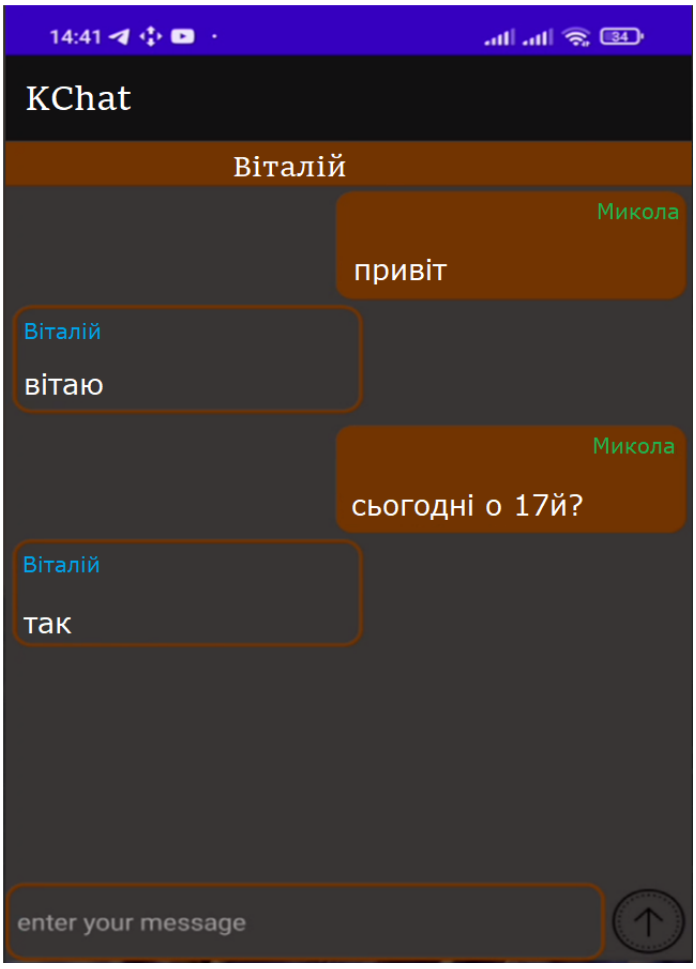

Рисунок 3.4 – Інтерфейс головного екрану системи опрацювання даних для спілкування у локальній мережі

Інтерфейс екрану авторизації наведено на рис. 3.5. Екран зі списком користувачів наведено на рис. 3.6.

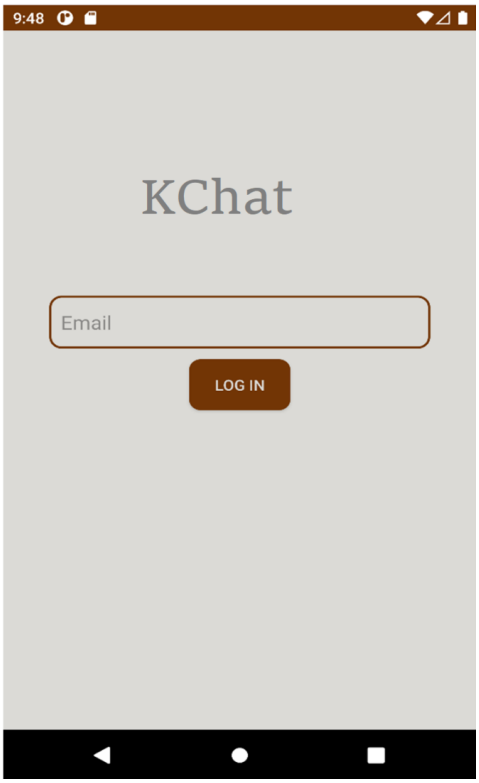

Рисунок 3.5 – Інтерфейс екрану авторизації

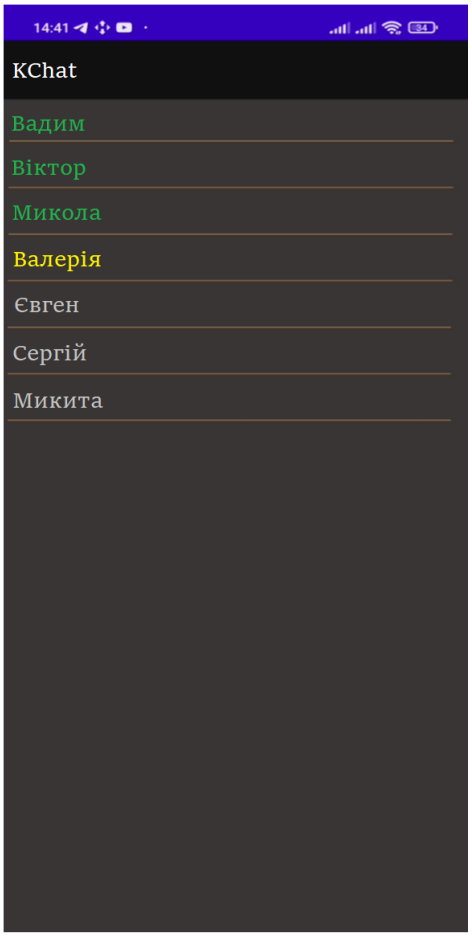

Рисунок 3.6 – Екран зі списком користувачів

3.5 Висновки за розділом 3

Розроблено систему опрацювання даних для спілкування у локальній мережі.

Запропоновано структуру системи опрацювання даних для спілкування у локальній мережі.

Визначено, що структура системи опрацювання даних для спілкування у локальній мережі передбачає логічне розділення її складових на три пакети: пакет app забезпечує взаємодію з користувачем та представленням інформації, що відображається на екрані мобільного пристрою; пакет data відповідає за роботу з даними та поєднує в собі реалізацію механізмів збереження та отримання інформації; пакет domain виконує функцію логічного посередника між джерелами даних і компонентами, які їх використовують. Кожен пакет виконує власну роль у забезпеченні коректного функціонування застосунку. Така організація сприяє підвищенню зручності розробки, підтримуваності та масштабованості системи, а також полегшує поділ відповідальності між програмними компонентами.

Описано функціонування системи опрацювання даних для спілкування у локальній мережі.

Розроблено базу даних системи опрацювання даних для спілкування у локальній мережі, яка відображає концептуальні зв'язки між обраними елементами у відповідній предметній області.

Логічна модель бази ґрунтується на фіксації користувачів локальної мережі та збереженні повідомлень, які пересилаються між ними. У структурі бази даних виділяються сутності, що відповідають за зберігання інформації про кожного користувача: його унікального ідентифікатора, відображуваного імені та стану активності у системі. Ще одна сутність описує повідомлення, які формуються та передаються в ході діалогу. Кожне повідомлення пов’язується з відправником і отримувачем за допомогою зовнішніх ключів,

що дозволяє чітко простежувати логіку спілкування між конкретними користувачами.

Виконано проєктування інтерфейсу взаємодії користувача з системою опрацювання даних для спілкування у локальній мережі.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ ОПРАЦЮВАННЯ ДАНИХ

4.1 Призначення й умови застосування системи опрацювання даних

Система опрацювання даних призначена для підтримки процесу спілкування у локальній мережі. Система опрацювання даних функціонує у операційній системі Android, написана на мові Kotlin, яка поєднує безпеку, високу продуктивність, зручність розробки і нативну підтримку Android, що робить цю мову оптимальним вибором для створення відповідної програми. В якості середовища розробки для створення системи опрацювання даних для спілкування у локальній мережі обрано Android Studio, яке є офіційним середовищем, створеним і підтримуваним Google, що гарантує повну сумісність із платформою та доступ до всіх актуальних інструментів, API і сервісів.

Система опрацювання даних для спілкування у локальній мережі забезпечує виконання таких функцій:

– підтримка можливості авторизації користувачів;

– підтримка можливості відображення переліку користувачів системи для спілкування;

– можливість відображення статусу користувачів (доступний до спілкування, недоступний для спілкування, офлайн);

– можливість спілкування з обраними користувачами за допомогою текстових повідомлень.

Для роботи системи опрацювання даних для спілкування у локальній мережі потрібні такі технічні та програмні ресурси: застосунок має функціонувати на мобільному пристрої з операційною системою Android, обладнаному процесором із тактовою частотою не нижче 1,5 ГГц для забезпечення стабільної роботи та оперативної обробки даних у реальному часі. Обсяг оперативної пам’яті пристрою повинен становити не менше 2 ГБ,

що гарантує коректне відображення елементів інтерфейсу й безперебійне виконання фонових процесів. Для зберігання локальної бази даних та інсталяції застосунку бажано мати не менше 200 МБ вільного простору у внутрішній пам’яті пристрою.

Мережеве підключення, необхідне для встановлення зв’язку між учасниками спілкування, може бути реалізоване через Wi-Fi або локальне з’єднання з безперебійним доступом до маршрутизатора або сервера, що обслуговує сокетні підключення. Стабільність роботи системи значною мірою залежить від пропускної здатності мережі та мінімальної затримки сигналу, тому бажано, щоб локальна мережа мала характеристики, достатні для обміну повідомленнями без відчутних затримок у передачі.

Програмне забезпечення пристрою повинно відповідати вимогам сучасних Android-компонентів, тож доцільним є використання версії операційної системи Android не нижче 8.0, що забезпечує сумісність із сучасними бібліотеками, такими як Room для управління базою даних та інструментами для роботи з сокетами. Для коректної роботи застосунку необхідне попереднє увімкнення мережевих дозволів та функцій бездротового зв’язку.

Серверна складова системи також має працювати на обладнанні з належною продуктивністю, здатному обробляти сокетні підключення кількох учасників одночасно. Вона повинна функціонувати під керуванням операційної системи, що підтримує відповідне мережеве програмне забезпечення, а також мати постійний доступ до локальної мережі.

Такий комплекс технічних і програмних вимог гарантує стабільне функціонування системи, своєчасну обробку обмінюваної інформації та комфортний для користувача процес спілкування в межах локальної мережі.

Функціональні характеристики розробленої системи опрацювання даних для спілкування у локальній мережі наведено у технічному завданні (додаток А). Фрагмент тексту програмного коду системи опрацювання даних для спілкування у локальній мережі наведено у додатку Б.

4.2 Характеристики системи опрацювання даних для спілкування у локальній мережі

Характеристики системи опрацювання даних для спілкування у локальній мережі визначають її здатність забезпечувати надійну, швидку та зручну взаємодію між користувачами. У процесі функціонування застосунку забезпечується стабільний обмін повідомленнями в режимі реального часу, без відчутних затримок та переривань, що особливо важливо під час комунікації в локальному середовищі. Система підтримує одночасне підключення кількох користувачів, дозволяючи кожному з них бачити актуальний список активних учасників і розпочинати діалог без потреби у складних діях або додаткових налаштуваннях.

Інтерфейс системи опрацювання даних для спілкування у локальній мережі реалізується таким чином, щоб забезпечити інтуїтивне керування та швидкий доступ до основних функцій. Увесь обмін даними здійснюється через централізований сервер або мережевий вузол, що відповідає за маршрутизацію повідомлень і підтримку з’єднання. Застосовується механізм автоматичного оновлення інформації, завдяки чому система постійно відображає актуальний стан мережевої активності. Усі дані, пов’язані зі спілкуванням, зберігаються локально в базі даних, що дозволяє переглядати історію обмінів навіть за відсутності підключення до мережі.

Система опрацювання даних для спілкування у локальній мережі розробляється з урахуванням вимог до захисту інформації: передбачено перевірку справності з’єднання, контроль доступу до мережевих ресурсів та уникнення некоректного завершення передачі даних. Архітектурна організація забезпечує простоту розширення й можливість інтеграції нових функціональних модулів у майбутньому, не порушуючи загального принципу роботи.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для запуску системи опрацювання даних для спілкування у локальній мережі на серверній частини необхідно запустити відповідний файл.

4.3.2 Вхідні й вихідні дані

Вхідними даними до системи опрацювання даних для спілкування у локальній мережі є інформація, яку користувач вводить для ідентифікації себе в мережі, зокрема ім’я, що відображатиметься іншим учасникам під час комунікації. Також до вхідних даних належать текстові повідомлення, які користувач надсилає співрозмовнику в процесі діалогу. Уся інформація, що надходить від користувача, обробляється системою та передається іншим підключеним учасникам відповідно до обраного напряму комунікації.

Вихідними даними системи опрацювання даних для спілкування у локальній мережі є отримані повідомлення, що надходять від інших користувачів, а також інформація про доступних для спілкування учасників, які на поточний момент під’єднанні до сервера. Система також генерує оновлені дані щодо стану мережевого з’єднання, які відображаються у вигляді зміни доступності співрозмовників або неможливості встановити зв’язок.

4.3.3 Повідомлення

Користувач системи опрацювання даних для спілкування у локальній мережі може отримати такі повідомлення: сповіщення про успішне підключення до мережі, попередження щодо відсутності зв’язку з Інтернетом або сервером, а також повідомлення про появу чи вихід інших учасників із мережі.

4.4 Виконання системи опрацювання даних для спілкування у локальній мережі

Після запуску системи опрацювання даних для спілкування у локальній мережі користувачу відображається екран авторизації (рис. 4.1), де необхідно ввести свою електронну адресу, натиснути кнопку Login, після цього ввести пароль.

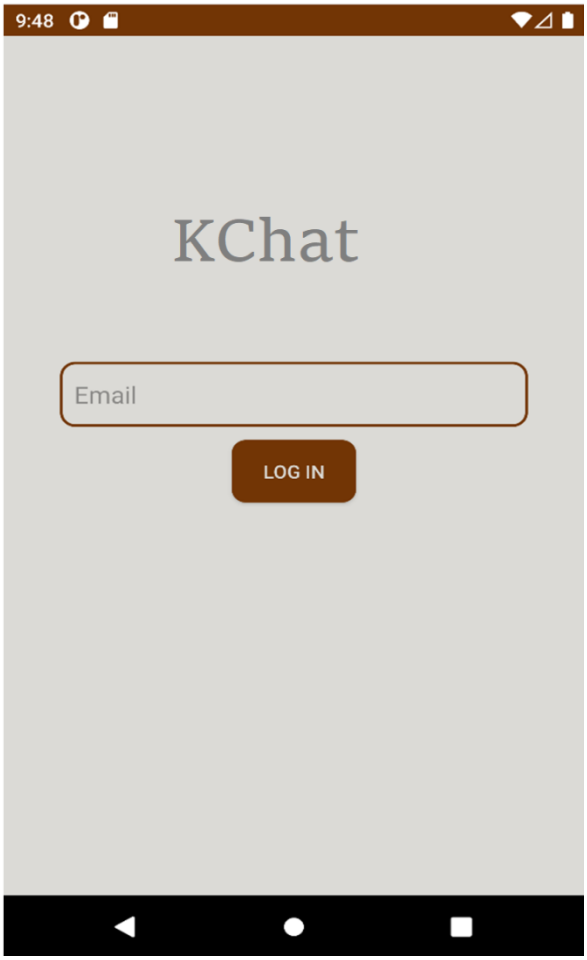

Рисунок 4.1 – Головний екран системи опрацювання даних для спілкування у локальній мережі

Після входу відображається список користувачів (рис. 4.2). При цьому зеленим кольором виділено активних користувачів, що є доступними для спілкування у поточний момент. Жовтим кольором виділено користувачів, які

знаходяться онлайн, але є недоступними для спілкування. Сірим кольором зображено користувачів, які у даний момент часу знаходяться офлайн.

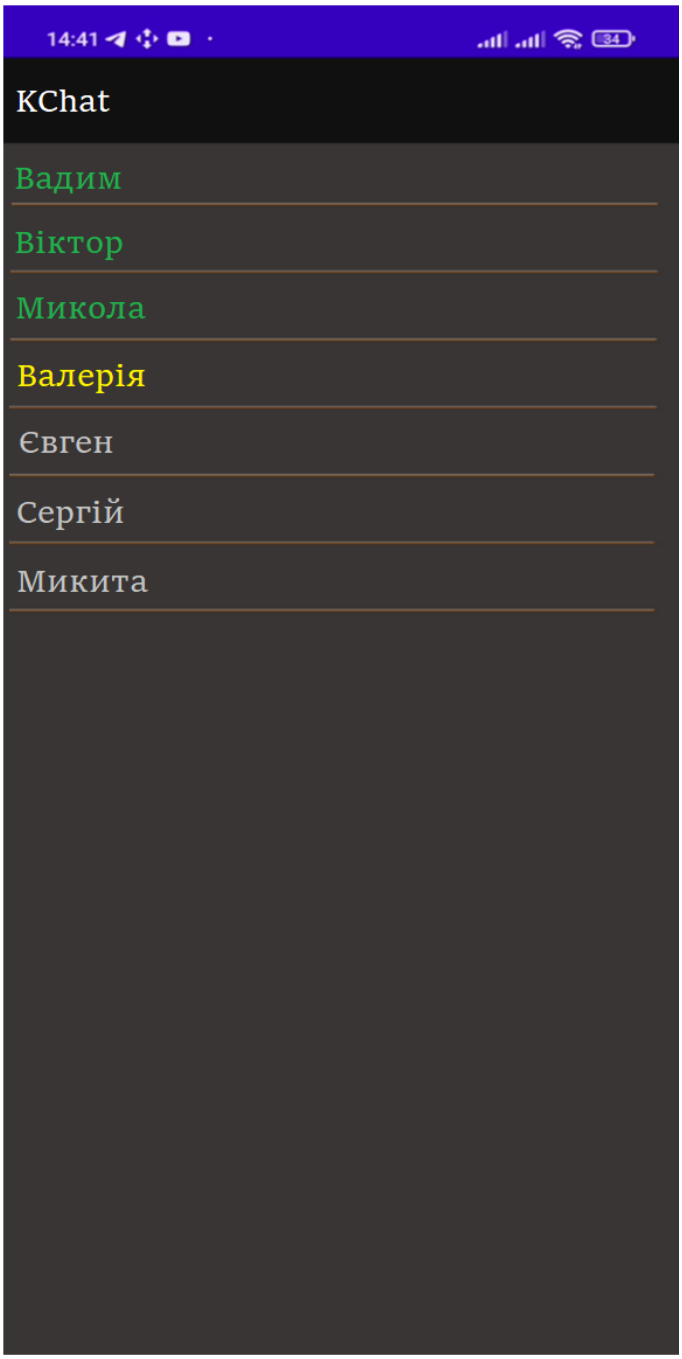

Рисунок 4.2 – Список користувачів

Для спілкування необхідно обрати потрібного активного користувача та відправити повідомлення. У вікні спілкування (рис. 4.3) відображаються надіслані та прийняті повідомлення від відповідних користувачів.

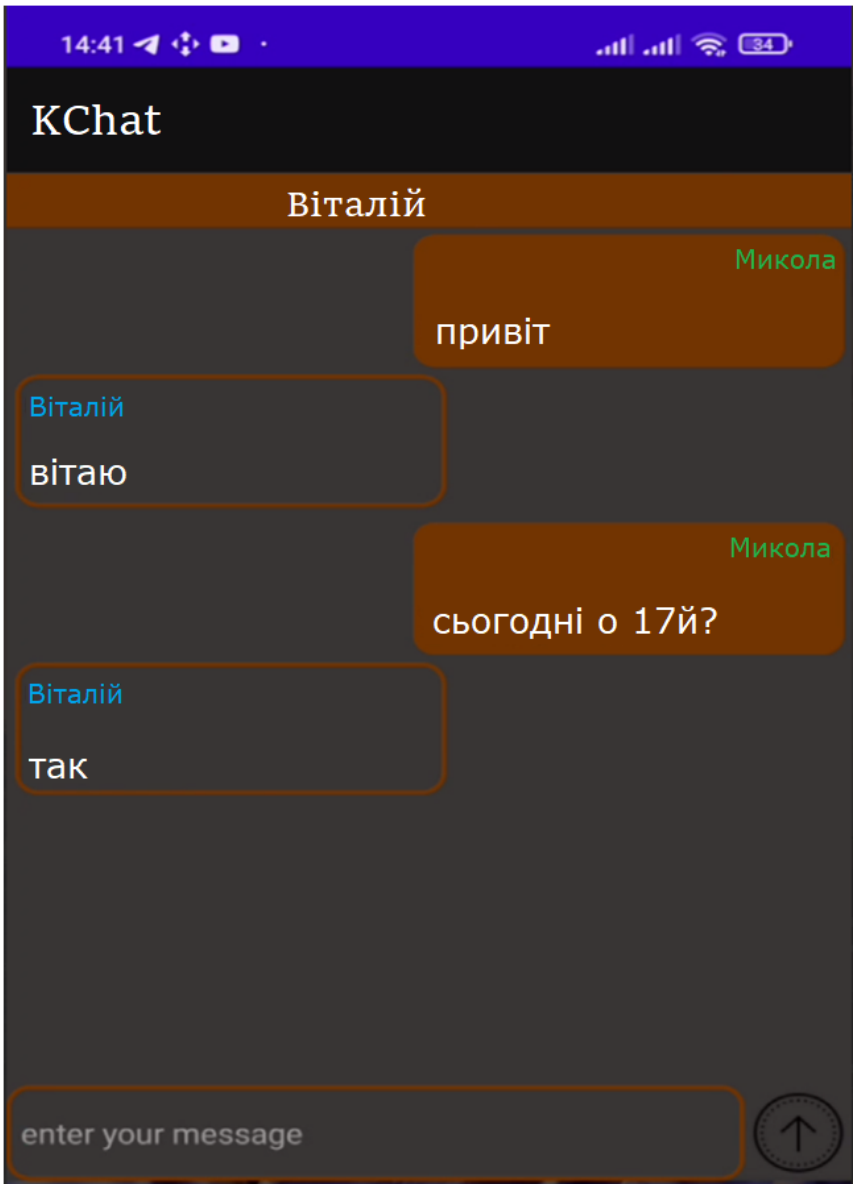

Рисунок 4.3 – Екран спілкування користувачів

4.5 Тестування системи опрацювання даних для спілкування у локальній мережі

Проведено тестування системи опрацювання даних для спілкування у локальній мережі.

Підтверджено, що системи опрацювання даних для спілкування у локальній мережі працює коректно і стабільно, виконуючи всі необхідні функціональні вимоги та успішно справляючись із основною задачею підтримки процесу взаємодії користувачів.

Розроблена система забезпечує організацію та опрацювання даних, пов’язаних із підтримкою процесів спілкування у локальній мережі.

4.6 Висновки за розділом 4

Описано систему опрацювання даних для спілкування у локальній мережі.

Проведено тестування створеної системи опрацювання даних для спілкування у локальній мережі. Результати тестування показали, що система забезпечує організацію та опрацювання даних, пов’язаних із підтримкою процесів спілкування у локальній мережі.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процеси обчислень, пов’язані з організацією та опрацюванням даних для обміну інформацією у локальній мережі.

Визначено, що система опрацювання даних для спілкування у локальній мережі забезпечує обмін повідомленнями між користувачами, під’єднаними до однієї мережі без використання зовнішніх серверів чи доступу до Інтернету. Така система обробляє, передає та зберігає дані у межах внутрішньої мережевої інфраструктури, забезпечуючи швидку та захищену комунікацію між пристроями.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато систем опрацювання даних для спілкування у локальній мережі. Проте деякі системи опрацювання даних для спілкування у локальній мережі є досить дорогими, крім того їх використання пов’язано з потребою в постійному технічному обслуговуванні мережевої інфраструктури та серверного обладнання. Відмова будь-якого елемента або збій в енергопостачанні можуть призвести до повної зупинки комунікації. Забезпечення належного рівня безпеки також вимагає професійного адміністрування, оскільки локальні системи, без оновлень і моніторингу, можуть стати вразливими до внутрішніх загроз. Крім того, масштабування такої системи опрацювання даних при зростанні кількості користувачів або географічному розширенні організації може потребувати значних матеріальних і часових витрат. Тому актуальною є розробка системи опрацювання даних для спілкування у локальній мережі.

Сформульовано функціональні вимоги до системи опрацювання даних для спілкування у локальній мережі.

Для функціонування системи опрацювання даних для спілкування у локальній мережі обрано операційну систему Android, яка забезпечує оптимальне поєднання доступності, функціональної розвиненості, мережевої

сумісності й економічної ефективності, що робить його раціональним вибором для реалізації системи опрацювання даних для спілкування у локальній мережі.

Для реалізації системи опрацювання даних для спілкування у локальній мережі обрано мову програмування Kotlin, яка поєднує безпеку, високу продуктивність, зручність розробки і нативну підтримку Android, що робить цю мову оптимальним вибором для створення відповідної програми.

Для створення системи опрацювання даних для спілкування у локальній мережі обрано середовище Android Studio, яке є офіційним середовищем, створеним і підтримуваним Google, що гарантує повну сумісність із платформою та доступ до всіх актуальних інструментів, API і сервісів. Це дозволяє розробникам максимально ефективно використовувати можливості мобільних пристроїв для організації локальної мережевої взаємодії. Крім того, Android Studio надає інструменти для створення, тестування та налагодження мережевих застосунків, а також має широку спільноту та великий обсяг навчальних ресурсів, що спрощує впровадження рішень і вирішення технічних проблем у процесі розробки.

Для зберігання та опрацювання даних обрано систему управління базами даних Room, яка поєднує повну інтеграцію з Android, підтримку Kotlin і сучасних архітектурних компонентів, високу продуктивність при роботі з локальними даними, асинхронність та надійність, а також зручні механізми управління схемами та міграціями. У результаті використання Room забезпечує легке створення, масштабування та підтримку стабільної системи опрацювання даних для локальної комунікації.

Розроблено систему опрацювання даних для спілкування у локальній мережі.

Запропоновано структуру системи опрацювання даних для спілкування у локальній мережі. Визначено, що структура системи опрацювання даних для спілкування у локальній мережі передбачає логічне розділення її складових на три пакети: пакет app забезпечує взаємодію з користувачем та представленням

інформації, що відображається на екрані мобільного пристрою; пакет data відповідає за роботу з даними та поєднує в собі реалізацію механізмів збереження та отримання інформації; пакет domain виконує функцію логічного посередника між джерелами даних і компонентами, які їх використовують. Кожен пакет виконує власну роль у забезпеченні коректного функціонування застосунку. Така організація сприяє підвищенню зручності розробки, підтримуваності та масштабованості системи, а також полегшує поділ відповідальності між програмними компонентами.

Описано функціонування системи опрацювання даних для спілкування у локальній мережі.

Розроблено базу даних системи опрацювання даних для спілкування у локальній мережі, яка відображає концептуальні зв'язки між обраними елементами у відповідній предметній області. Логічна модель бази ґрунтується на фіксації користувачів локальної мережі та збереженні повідомлень, які пересилаються між ними. У структурі бази даних виділяються сутності, що відповідають за зберігання інформації про кожного користувача: його унікального ідентифікатора, відображуваного імені та стану активності у системі.

Виконано проєктування інтерфейсу взаємодії користувача з системою опрацювання даних для спілкування у локальній мережі.

Проведено тестування створеної системи опрацювання даних для спілкування у локальній мережі. Результати тестування показали, що система забезпечує організацію та опрацювання даних, пов’язаних із підтримкою процесів спілкування у локальній мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Asynchronous Communication: What it is and How to Nail it. URL: https: //project.co/asynchronous-communication/ (date of access: 19.04.2026).

2. Messenger. URL: https: //contentstudio.io/social-media-terms/messenger (date of access: 19.04.2026).

3. Real-Time Chat. URL: https: //getstream.io/glossary/real-time-chat/ (date of access: 19.04.2026).

4. What Is Messenger. URL: https: //www.socialpilot.co/social-media-terms/messenger (date of access: 19.04.2026).

5. Live Chat: Overview, Benefits & Best Practices. URL: https: //blog.emb.global/what-is-live-chat/ (date of access: 19.04.2026).

6. Conversational Guide to Online Chat. URL: https: //www.chatomiser.com/guides/online-chat (date of access: 19.04.2026).

7. What Is Omnichannel Communication. URL: https: //learn.g2.com/omnichannel-communication (date of access: 19.04.2026).

8. Open-source Free LAN Messengers and Local Network Chat Clients. URL: https: //medevel.com/12-os-network-messangers/ (date of access: 19.04.2026).

9. Open Source LAN Messengers. URL: https: //sourceforge.net/directory/lan-messengers/ (date of access: 19.04.2026).

10. Most Popular Messaging Apps. URL: https: //explodingtopics.com/blog/messaging-apps-stats (date of access: 19.04.2026).

11. Top 10 Best message apps for Windows. URL: https: //rambox.app/blog/best-message-apps-for-windows/ (date of access: 19.04.2026).

12. Mesh a Secure, Anonymous, Peer-to-peer Instant Messenger. URL: https: //mesh.im/ (date of access: 19.04.2026).

13. KouChat: Serverless LAN Chat. URL: https: //www.kouchat.net/ (date of access: 19.04.2026).

14. Khernet Chat instantly, No internet, No servers. URL: https: //www.khernet.app/ (date of access: 19.04.2026).

15. Android Application Components. URL: https: //www.tutorialspoint.com/android/android_application_components.htm (date of access: 19.04.2026).

16. Android platform. URL: https: //developer.android.com/about/ (date of access: 19.04.2026).

17. Kotlin Data Types. URL: https: //www.w3schools.com/kotlin/kotlin_data_types.php (date of access: 19.04.2026).

18. Develop Android apps with Kotlin. URL: https: //developer.android.com/kotlin/ (date of access: 19.04.2026).

19. New UI in Android Studio. URL: https: //developer.android.com/studio/intro/new-ui (date of access: 19.04.2026).

20. How To Install Android Studio. URL: https: //tms-outsource.com/blog/posts/how-to-install-android-studio/ (date of access: 19.04.2026).

21. Save data in a local database using Room. URL: https: //developer.android.com/training/data-storage/room/ (date of access: 19.04.2026).

22. Step-by-Step: Setting Up and Implementing Room Database in Android. URL: https: //medium.com/@sdranju/step-by-step-how-to-setting-up-and-implementing-room-database-aeb211c56702 (date of access: 19.04.2026).

ДОДАТОК А

Технічне завдання

Вступ

Система опрацювання даних може використовуватися для підтримки процесу спілкування у локальній мережі.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Створення програмної системи опрацювання даних для спілкування у локальній мережі», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 07 квітня 2026 р.

А.2 Призначення розробки

Система опрацювання даних призначена для забезпечення спілкування у локальній мережі.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Система опрацювання даних для спілкування у локальній мережі забезпечує виконання таких функцій:

– підтримка можливості авторизації користувачів;

– підтримка можливості відображення переліку користувачів системи для спілкування;

– можливість відображення статусу користувачів (доступний до спілкування, недоступний для спілкування, офлайн);

– можливість спілкування з обраними користувачами за допомогою текстових повідомлень.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс системи опрацювання даних для спілкування у локальній мережі повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному режимі.

А.3.3 Вимоги до надійності

Система опрацювання даних для спілкування у локальній мережі повинна забезпечити надійне функціонування.

А.3.4 Умови експлуатації

Для експлуатації системи опрацювання даних для спілкування у локальній мережі необхідна наявність персонального комп’ютера.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для роботи системи опрацювання даних для спілкування у локальній мережі потрібні такі технічні та програмні ресурси: застосунок має функціонувати на мобільному пристрої з операційною системою Android, обладнаному процесором із тактовою частотою не нижче 1,5 ГГц для забезпечення стабільної роботи та оперативної обробки даних у реальному часі. Обсяг оперативної пам’яті пристрою повинен становити не менше 2 ГБ, що гарантує коректне відображення елементів інтерфейсу й безперебійне виконання фонових процесів. Для зберігання локальної бази даних та інсталяції застосунку бажано мати не менше 200 МБ вільного простору у внутрішній пам’яті пристрою.

А.3.6 Вимоги до маркування і пакування

Система опрацювання даних для спілкування у локальній мережі може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва – «Система опрацювання даних для спілкування у локальній мережі».

А.4 Вхідні дані до роботи

Вхідними даними до системи опрацювання даних для спілкування у локальній мережі є інформація, яку користувач вводить для ідентифікації себе в мережі, зокрема ім’я, що відображатиметься іншим учасникам під час комунікації. Також до вхідних даних належать текстові повідомлення, які користувач надсилає співрозмовнику в процесі діалогу. Уся інформація, що надходить від користувача, обробляється системою та передається іншим підключеним учасникам відповідно до обраного напряму комунікації.

Вихідними даними системи опрацювання даних для спілкування у локальній мережі є отримані повідомлення, що надходять від інших користувачів, а також інформація про доступних для спілкування учасників, які на поточний момент під’єднанні до сервера. Система також генерує оновлені дані щодо стану мережевого з’єднання, які відображаються у вигляді зміни доступності співрозмовників або неможливості встановити зв’язок.

ДОДАТОК Б

Фрагмент тексту програми

```
class ChtFrgmnt : Frgmnt() {

    private lateinit var bndng: FrgmntChtBndng
    private val model by viewModel<ChtViewModel>(parameters = {
parametersOf(itmKrst) })

    private val itmKrst: UiKrst by lazy {
        requireArguments().getParcelable(KRSTRECVR)!!
    }

    private val adaptr by lazy {
        ChtAdaptr(
            recvr = itmKrst,
            you = model.getYou()
        )
    }

    override fun onCreateView(
        inflatr: LayoutInflatr,
        containr: ViewGroup?,
        savedInstansState: Bundle?
    ): View {
        bndng = FrgmntChtBndng.inflate(inflatr, containr, false)
        return bndng.root
    }

    override fun onViewCreated(view: View, savedInstansState:
Bundle?) {
        super.onViewCreated(view, savedInstansState)

        bndngReclr()

        with(bndng) {
            Krstname.text = itmKrst.name
            btnSndMssg.setOnClickLstnr {
                val mssg: String = editMssg.text.toString()
                model.sendMssg(mssg)
                editMssg.text.clear()
            }
        }
        obsrvViewModel()
    }

    private fun obsrvViewModel() {
        model.mssgs.obsrv(viewLifecycleOwnr, {
            adaptr.submitList(it ?: listOf())
        })
```

```
        model.erSrvr.obsrv(viewLifecycleOwnr, {
            Tst.makeText(
                activity?.applicationContext,
                TST_TXT_CONNECTION_LOST,
                Tst.LENGTH_SHORT
            ).show()
            transToTheFirstWindow()
        })
    }

    private fun transToTheFirstWindow() {
        activity?.supportFrgmntManager?.popBackStack("", 0)
        val Frgmnt = LgnFrgmnt()
        val FrgmntTransaction =
activity?.supportFrgmntManager?.beginTransaction()
        FrgmntTransaction?.replace(R.id.Frgmnt_containr, Frgmnt)
            ?.addToBackStack("")
            ?.commit()
    }

    private fun bndngReclr() {
        bndng.ReclrMssgs.layoutManager =
LinearLayoutManager(context)
        bndng.ReclrMssgs.adaptr = adaptr
    }

    companion object {
        private const val KRSTRECVR = "Krst"

        fun newInstans(recvr: Krst): ChtFrgmnt {
            return ChtFrgmnt().apply {
                arguments = Bundle().apply {
                    putParcelable(KRSTRECVR, recvr.toUi())
                }
            }
        }
    }
}


class ChtViewModel(
    private val Krst: UiKrst,
    private val rpstr: MssgsRpstr
) : ViewModel() {

    private val _mssgs: MutableLvDt<List<Mssg>> = MutableLvDt()
    val mssgs: LvDt<List<Mssg>> = _mssgs
```

```
    private val _erSrvr = SingleLiveEvent<Int>()
    val erSrvr: LvDt<Int> = _erSrvr

    init {
        viewModelSkp.launch {
            rpstr.getMssgsByKrstId(Krst.id).collect {
                _mssgs.value = it.map {
                    it.toDatabase()
                }
            }
        }
        viewModelSkp.launch {
            rpstr.gtEr().collect {
                withContext(Dispatchers.Main) {
                    _erSrvr.value = it
                }
            }
        }
    }

    fun sendMssg(mssg: String) {
        viewModelSkp.launch(Dispatchers.IO) {
            rpstr.sendMssg(Krst.toKrst(), mssg)
        }
    }

    fun getYou(): UiKrst {
        return rpstr.getYou().toUi()
    }

}

class LgnFrgmnt : Frgmnt() {

    private lateinit var bndng: FrgmntLgnBndng
    private val model by viewModel<LgnViewModel>()

    override fun onCreateView(
        inflatr: LayoutInflatr,
        containr: ViewGroup?,
        savedInstansState: Bundle?
    ): View {
        bndng = FrgmntLgnBndng.inflate(inflatr, containr, false)
        return bndng.root
    }

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

```
        super.onViewCreated(view, savedInstansState)

        obsrvToViewModel()
        with(bndng) {
            lgn.setOnClickLstnr {
                ProgresBar.visibility = ProgresBar.VISIBLE
                val name: String = editmail.text.toString()
                model.gtIp(name)
            }
        }
    }

    private fun transToTheNextWindow() {
        val Frgmnt = KrstsFrgmnt()
        val FrgmntTransaction =
activity?.supportFrgmntManager?.beginTransaction()
        FrgmntTransaction?.replace(
            R.id.Frgmnt_containr,
            Frgmnt
        )
            ?.addToBackStack("")
            ?.commit()
    }

    private fun obsrvToViewModel() {
        model.trigrNextNavigation.obsrv(this, {
            transToTheNextWindow()
        })
        model.erSrvr.obsrv(this, {
            bndng.ProgresBar.visibility = ProgresBar.INVISIBLE
            Tst.makeText(
                activity?.applicationContext,
                TST_TXT,
                Tst.LENGTH_SHORT
            ).show()
        })
    }
}

class LgnViewModel(
    private val udp: UdpClnt,
    private val tsp: TspClnt
) : ViewModel() {

    private val _trigrNextNavigation = SingleLiveEvent<Unit>()
    val trigrNextNavigation: LvDt<Unit> = _trigrNextNavigation

    private val _erSrvr = SingleLiveEvent<Unit>()
```

```
    val erSrvr: LvDt<Unit> = _erSrvr

    fun gtIp(name: String) {
        viewModelSkp.launch(Dispatchers.IO) {
            try {
                val ip = udp.getServerIp()
                tsp.createSckt(ip, name)
                withContext(Dispatchers.Main) {
                    _trigrNextNavigation.call()
                }
            } catch (e: Exception) {
                e.printStackTrace()
                withContext(Dispatchers.Main) {
                    _erSrvr.call()
                }
            }
        }
    }
}


class KrstsAdaptr(
    private val onKrstsClicked: (Krst) -> Unit
) : ListAdaptr<Krst, KrstsAdaptr.MyViewHolder>(DifCallback()) {

    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): MyViewHolder {
        val bndng = KrstLayoutBndng.inflate(
            LayoutInflatr.from(parent.context),
            parent,
            false
        )
        return MyViewHolder(bndng)
    }

    class MyViewHolder(private val bndng: KrstLayoutBndng) :
        ReclrView.ViewHolder(bndng.root) {

        fun bind(Krst: Krst, onClick: (Krst) -> Unit) {
            with(bndng) {
                KrstName.text = Krst.name
                root.setOnClickLstnr {
                    onClick(Krst)
                }
            }
        }
    }
```

```
    class DifCallback : DifUtil.ItmCallback<Krst>() {
        override fun areItmsTheSame(oldKrst: Krst, newKrst:
Krst): Boolean {
            return oldKrst.id == newKrst.id
        }

        override fun areContentsTheSame(oldKrst: Krst, newKrst:
Krst): Boolean {
            return oldKrst == newKrst
        }
    }

}

class KrstsFrgmnt : Frgmnt() {
    private lateinit var bndng: FrgmntKrstsBndng
    private val model by viewModel<KrstsViewModel>()
    private val adaptr by lazy {
        KrstsAdaptr(
            onKrstsClicked = {
                transToTheNextWindow(it)
            }
        )
    }

    override fun onCreateView(
        inflatr: LayoutInflatr,
        containr: ViewGroup?,
        savedInstansState: Bundle?
    ): View {
        bndng = FrgmntKrstsBndng.inflate(inflatr, containr,
false)
        return bndng.root
    }

    override fun onViewCreated(view: View, savedInstansState:
Bundle?) {
        super.onViewCreated(view, savedInstansState)

        bndngReclr()
        obsrvToViewModel()
        model.getKrsts()
    }

    private fun bndngReclr() {
        bndng.Reclr.layoutManager = LinearLayoutManager(context)
        bndng.Reclr.adaptr = adaptr
    }
```

```
    private fun transToTheNextWindow(Krst: Krst) {
        val Frgmnt = ChtFrgmnt.newInstans(Krst)
        val FrgmntTransaction =
activity?.supportFrgmntManager?.beginTransaction()
        FrgmntTransaction?.replace(
            R.id.Frgmnt_containr,
            Frgmnt
        )
            ?.addToBackStack("")
            ?.commit()
    }

    private fun obsrvToViewModel() {
        model.Krsts.obsrv(viewLifecycleOwnr, { t ->
            adaptr.submitList(t ?: listOf())
        })
        model.erSrvr.obsrv(viewLifecycleOwnr, {
            Tst.makeText(
                activity?.applicationContext,
                TST_TXT_CONNECTION_LOST,
                Tst.LENGTH_SHORT
            ).show()
            transToTheFirstWindow()
        })
    }

    private fun transToTheFirstWindow() {
        val Frgmnt = LgnFrgmnt()
        val FrgmntTransaction =
activity?.supportFrgmntManager?.beginTransaction()
        FrgmntTransaction?.replace(R.id.Frgmnt_containr, Frgmnt)
            ?.addToBackStack("")
            ?.commit()
    }
}

class KrstsViewModel(
    private val tsp: TspClnt
) : ViewModel() {

    private val _Krsts: MutableLvDt<List<Krst>> = MutableLvDt()
    val Krsts: LvDt<List<Krst>> = _Krsts

    private val _erSrvr = SingleLiveEvent<Int>()
    val erSrvr: LvDt<Int> = _erSrvr

    init {
```

```
        obsrvr()
        getKrsts()
    }

    fun getKrsts() {
        viewModelSkp.launch(Dispatchers.IO) {
            while (true) {
                delay(DELAY)
                tsp.getKrsts()
            }
        }
    }

    private fun obsrvr() {
        viewModelSkp.launch {
            val shwErr = tsp.gtEr()
            shwErr.collect {
                withContext(Dispatchers.Main) {
                    _erSrvr.value = it
                }
            }

        }
        viewModelSkp.launch {
            val KrstsList = tsp.getKrstsList()
            KrstsList.collect {
                _Krsts.value = it.Krsts
            }
        }
    }
}

internal class TspClntImpl : TspClnt {
    private fun png() {
        skp.launch {
            while (!sckt.isClosed) {
                delay(DELAY)
                try {
                    val png = PngDto(
                        you.value!!.id
                    ).toJson()
                    send(png)
                    timer = skp.launch {
                        delay(DELAY)
                        close()
                    }
                } catch (e: Exception) {
                    e.printStackTrace()
```

```
                    close()
                }
            }
        }
    }

    private fun startMssgLoop() {
        skp.launch {
            while (!sckt.isClosed) {
                try {
                    val bsDto = gsn.fromJson(rdr.readLine(),
BsDto::class.java)
                    when (bsDto.action) {
                        BsDto.Action.CONNECTED -> {
                            KrstId.value =
gsn.fromJson(bsDto.payload, ConnectedDto::class.java)
                            confirmConnect()
                        }
                        BsDto.Action.PONG -> {
                            timer?.cancel()
                        }
                        BsDto.Action.KRSTS_RECEIVED -> {
                            KrstsList.emit(
                                gsn.fromJson(
                                    bsDto.payload,
                                    KrstsReceivedDto::class.java
                                )
                            )
                        }
                        BsDto.Action.NEW_MSSG -> {
                            newMssg.value = gsn.fromJson(
                                bsDto.payload,
                                MssgDto::class.java
                            )
                        }
                        else -> {
                            close()
                        }
                    }
                } catch (e: java.lang.Exception) {
                    shwErr.emit(ERROR404)
                    e.printStackTrace()
                    close()
                }
            }
        }
    }
```

```
private suspend fun confirmConnect() {
    val id: String = KrstId.first()?.id ?: ""
    val name = KrstName.first() ?: ""
    you.emit(Krst(id, name))
    val connect = ConnectDto(
        id = id,
        name = name
    ).toJson()
    send(connect)
    png()
}

override suspend fun getKrsts() {
    val KrstId = you.first()?.id ?: ""
    val getKrsts = GetKrstsDto(
        KrstId
    ).toJson()
    send(getKrsts)
}

override fun getKrstsList(): Flow<KrstsReceivedDto> {
    return KrstsList
}

override suspend fun sendMssg(recvr: String, mssg: String) {
    val sendMssg = SendMssgDto(
        you.value!!.id,
        recvr,
        mssg
    ).toJson()
    send(sendMssg)
}

override fun getNewMssg(): Flow<MssgDto> {
    return newMssg.filterNotNull()
}

override fun gtEr(): Flow<Int> {
    return shwErr
}

override fun getYou(): Krst {
    return you.value!!
}

private fun send(mssg: String) {
    wrtr.println(mssg)
    wrtr.flush()
```

```
    }

    private fun close() {
        wrtr.close()
        rdr.close()
        sckt.close()
        job.cancelChildren()
    }

}
internal class UdpClntImpl : UdpClnt {

    override suspend fun getServerIp(): String {
        val sckt = DatagramSckt()
        var ip = ""
        var klk = KLK

        while (ip.isEmpty()) {

            if (klk == 0) {
                throw NoConnectionPendingException()
            }
            klk--

            try {
                val mssg = MSSG.toByteArray()
                var packet = DatagramPacket(
                    mssg,
                    mssg.size,
                    InetAddress.getByName(HOST),
                    UPD_PORT
                )
                sckt.soTimeout = DELAY.toInt()
                sckt.send(packet)

                val buffer = ByteArray(BYTEARRAYSIZE)
                packet = DatagramPacket(buffer, buffer.size)
                sckt.receive(packet)
                ip = packet.address.hostAddress

            } catch (e: ScktException) {
                e.printStackTrace()
            } catch (e: IOException) {
                e.printStackTrace()
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }
```

```
        sckt.close()
        return ip
    }
}

internal class MssgsRpstrImpl(
    private val tsp: TspClnt,
    private val localRpstr: LocalRpstr,
    private val idGenerator: IdGenerator
) : MssgsRpstr {

    private val mssg = tsp.getNewMssg()
    private val you = tsp.getYou()

    private val job = SupervisorJob()
    private val skp = CoroutineSkp(Dispatchers.IO + job)

    private val errorLstnr = MutableSharedFlow<Int>()

    init {
        skp.launch(Dispatchers.IO) {
            localRpstr.deleteAllChts()
            mssg.collect {
                localRpstr.addMssg(
                    mssg = DomainMssg(
                        id = idGenerator.generateId(),
                        from = it.from.id,
                        to = you.id,
                        mssg = it.mssg
                    )
                )
            }
        }
        skp.launch {
            val gtEr = tsp.gtEr()
            gtEr.collect {
                errorLstnr.emit(it)
            }
        }
    }

    override suspend fun getMssgsByKrstId(id: String):
Flow<List<DomainMssg>> {
        return localRpstr.getMssgById(id)
    }

    override suspend fun sendMssg(Krst: Krst, mssg: String) {
```

```
            tsp.sendMssg(Krst.id, mssg)
            val newMssg = DomainMssg(
                id = idGenerator.generateId(),
                from = you.id,
                to = Krst.id,
                mssg = mssg
            )
            localRpstr.addMssg(newMssg)
        }

        override fun getYou(): Krst {
            return you
        }

        override fun gtEr(): Flow<Int> {
            return errorLstnr
        }
    }
```

ДОДАТОК В

Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Створення програмної системи опрацювання даних для спілкування у локальній мережі

Виконав	Віталій КУЛИКОВ ст.гр. КНТ-213сп
Керівник	Тетяна КОЛПАКОВА к.т.н., доцент

Рисунок В.1 – Слайд 1

2

Об’єкт, предмет та мета роботи

Об’єкт дослідження – процеси обчислень, пов’язані з організацією та опрацюванням даних для обміну інформацією у локальній мережі.

Предмет дослідження – програмні системи опрацювання даних для спілкування у локальній мережі.

Мета роботи – розробка програмної системи опрацювання даних для спілкування у локальній мережі для підвищення ефективності взаємодії користувачів.

Рисунок В.2 – Слайд 2

3

Задачі роботи

– виконати аналіз предметної області та систем опрацювання даних для спілкування у локальній мережі;

– здійснити проєктування системи опрацювання даних для спілкування у локальній мережі;

– створити систему опрацювання даних для спілкування у локальній мережі;

– дослідити розроблену систему опрацювання даних для спілкування у локальній мережі.

Рисунок В.3 – Слайд 3

4

Порівняння існуючих аналогів

Критерій порівняння	Mesh	KouChat	Khernet
Простота інсталяції та запуску	+	+–	+
Портативність	+–	+–	+
Шифрування повідомлень	+	–	+
Підтримка кількох платформ	+	+	+–
Запис історії чату	+	+	+–
Можливість змінювати тему/інтерфейс	–	+	–

Рисунок В.4 – Слайд 4

5

Постановка завдання

При розробці системи опрацювання даних для спілкування у локальній мережі необхідно забезпечити такі функціональні вимоги:

– підтримка можливості авторизації користувачів;

– підтримка можливості відображення переліку користувачів системи для спілкування;

– можливість відображення статусу користувачів (доступний до спілкування, недоступний для спілкування, офлайн);

– можливість спілкування з обраними користувачами за допомогою текстових повідомлень.

Рисунок В.5 – Слайд 5

6

Вибір мови програмування

Критерій порівняння мов програмування	Мова програмування		
	Kotlin	Java	C#
Нативна підтримка Android	+	+–	–
Підтримка асинхронності та багатопоточності	+	+–	+
Безпека типів	+	+–	+–
Продуктивність при роботі з мобільними мережевими API	+	+	+–
Можливість швидкої розробки та тестування	+	+–	+–

Рисунок В.6 – Слайд 6

7

Вибір середовища розробки

Критерій порівняння середовищ розробки	Середовища розробки		
	Android Studio	IntelliJ IDEA	Eclipse
Офіційна підтримка Android та Kotlin	+	+–	–
Вбудовані емулятори та мережеве тестування	+	+–	–
Можливість комплексного налагодження застосунків	+	+	+–
Інструменти аналізу продуктивності та ресурсів	+	+–	+–
Простота налаштування середовища	+	+–	+–

Рисунок В.7 – Слайд 7

8

Вибір засобів зберігання та опрацювання даних

Критерій порівняння систем управління базами даних	Системи управління базами даних		
	Room	SQLite	Realm
Простота роботи з об'єктами даних	+	–	+
Контроль схем і міграцій	+	+–	+–
Стабільність та надійність для локальних даних	+	+	+–
Продуктивність при роботі з великими обсягами локальних даних	+	+–	+
Простота інтеграції з архітектурними патернами Android	+	–	+–

Рисунок В.8 – Слайд 8

9

Структура системи опрацювання даних для спілкування у локальній мережі

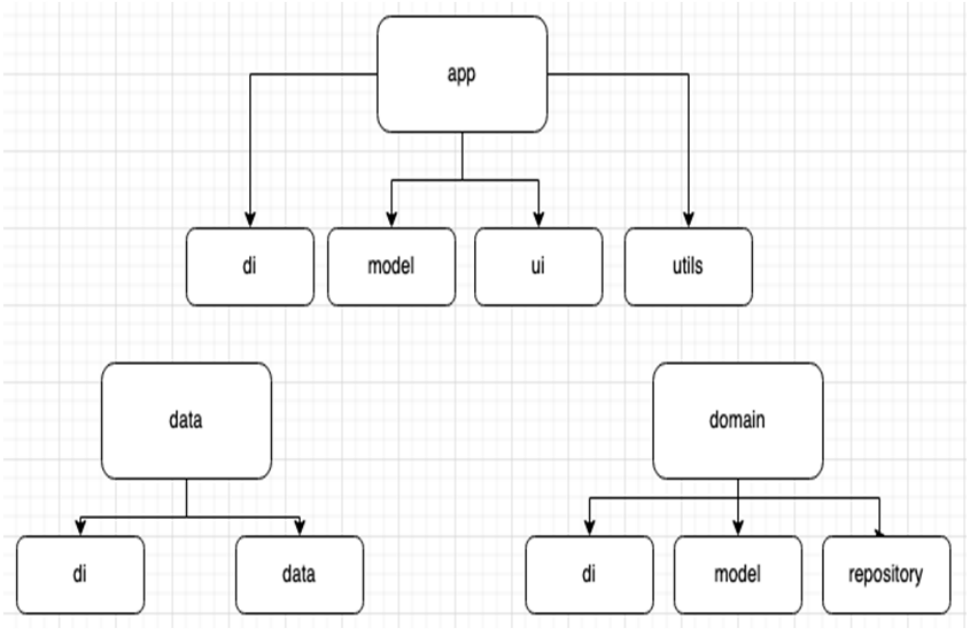

Рисунок В.9 – Слайд 9

10

Схема функціонування системи опрацювання даних для спілкування у локальній мережі

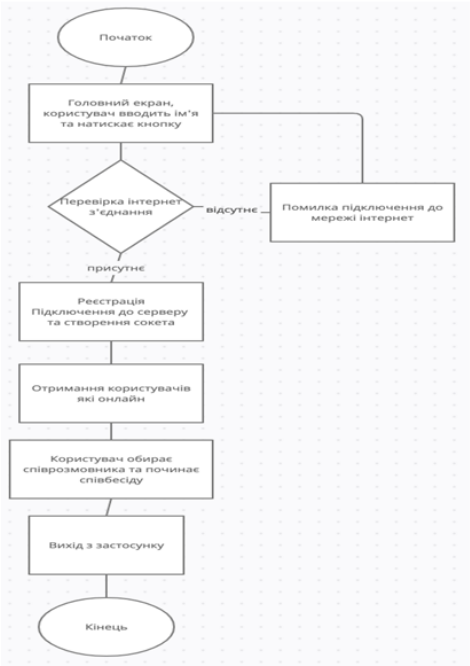

Рисунок В.10 – Слайд 10

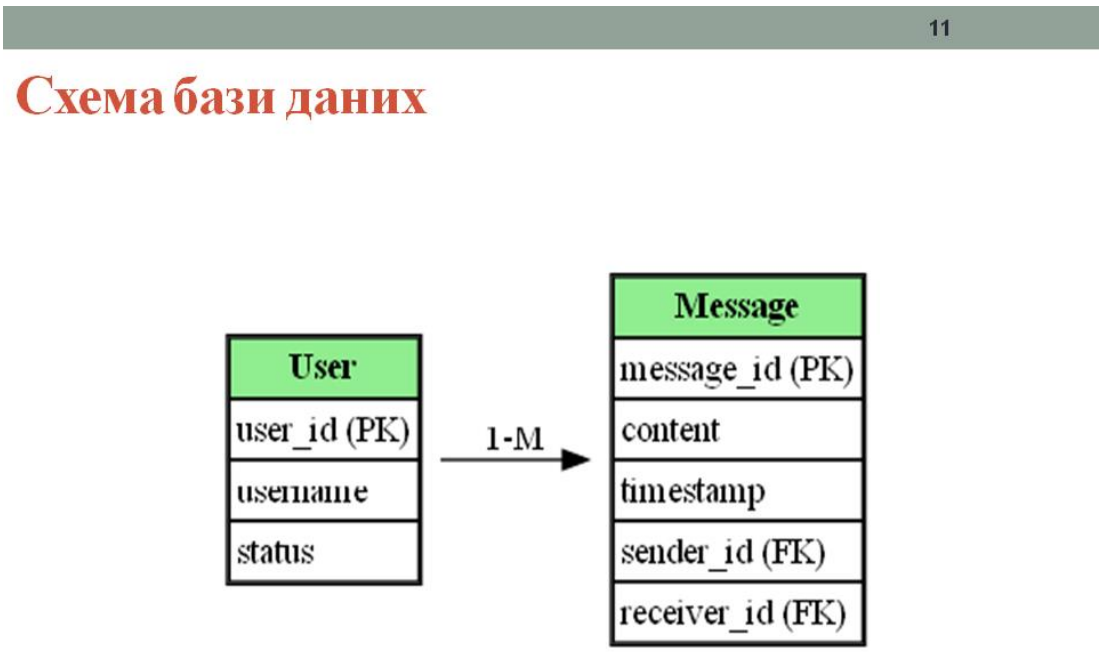

Рисунок В.11 – Слайд 11

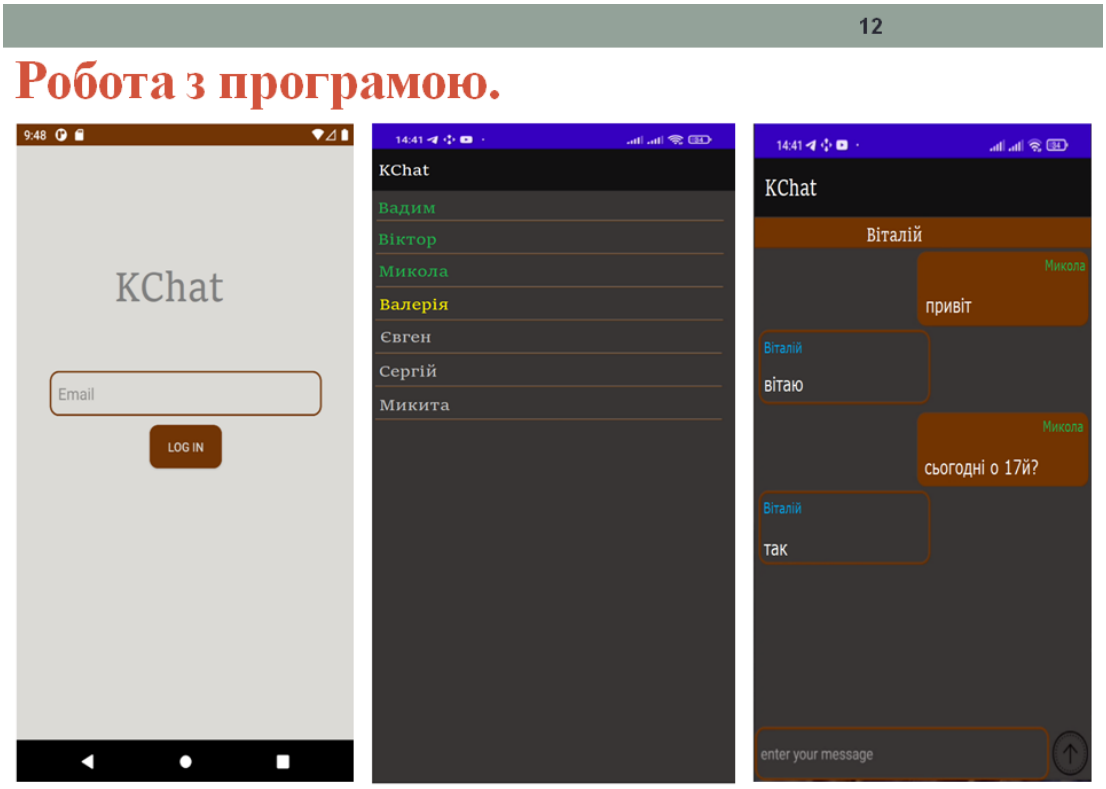

Рисунок В.12 – Слайд 12

Висновки

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процеси обчислень, пов'язані з організацією та опрацюванням даних для обміну інформацією у локальній мережі.

Досліджено системи опрацювання даних для спілкування у локальній мережі. Визначено функціональні вимоги до системи опрацювання даних для спілкування у локальній мережі.

Для створення системи опрацювання даних для спілкування у локальній мережі обрано мову програмування Kotlin, середовище Android Studio та систему управління базами даних Room.

Розроблено систему опрацювання даних для спілкування у локальній мережі .

Виконано тестування розробленої системи опрацювання даних для спілкування у локальній мережі .

Усі завдання дипломної кваліфікаційної роботи бакалавра повністю виконано.

Рисунок В.13 – Слайд 13