

ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики та інформатики

«До захисту допущено»

Завідувач кафедри ПМІ

Н.О. Маслова

(підпис)

(ініціали, прізвище)

« _____ » _____ 2024 р.

Випускна кваліфікаційна робота
бакалавра

(освітньо-кваліфікаційний рівень)

на тему

«Анонімні соціальні чати»

зі спецчастиною

«Розробка модулю підтримки зв'язку між

співрозмовниками для анонімного чату»

Виконав: студент 4 курсу, групи 121

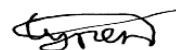
(шифр групи)

спеціальності 121 Інженерія програмного забезпечення

(шифр і назва напрямку підготовки, спеціальності)

Погосян Сурен Мелікович

(прізвище та ініціали)



(підпис)

Керівник проф. кафедри ПМІ, д.т.н., проф. Башков Є.О.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

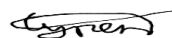
Рецензент проф. каф. ПЗКС НТУ "Дніпровська політехніка",

д.т.н., доц. Іван Лактіонов

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

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



Студент _____

(підпис)

Луцьк – 2024

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступінь бакалавр
Напрямок підготовки (спеціальність) 121 « »

ЗАТВЕРДЖУЮ:

Завідувачка кафедри ПМІ

Н.О. Маслова / _____ /

“ ____ ” _____ 2024 р.

ЗАВДАННЯ НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Погосян Сурен Мелікович

(прізвище, ім'я, по батькові)

1. Тема роботи «Анонімні чати»

Спецчастина «Розробка модулю підтримки зв'язку між співрозмовниками
для анонімного чату»

Керівник роботи Башков Є. О., д.т.н., проф., проф. каф. ПМІ

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від “ 25 ” квітня 2024 року № XXX

2. Строк подання студентом роботи “03” червня 2024 року

3. Вихідні дані до роботи результати науково-дослідної роботи, матеріали
переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

1) аналіз предметної області та постановка задачі;

2) огляд та порівняння сучасних анонімних чатів

3) розробка модулю підтримки зв'язку між співрозмовниками для
анонімного чату.

4) розробка модулю для анонімного спілкування між людьми за
допомогою React Native та Node.js

5) тестування розробленого програмного модуля;

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) робота повинна містити рисунки та інший графічний матеріал, у кількості, достатній для демонстрації результатів кваліфікаційної роботи

6. Консультанти розділів роботи

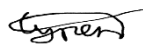
Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтролер	доц. Назарова І.А.		

7. Дата видачі завдання 24 квітня 2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Прим.
1	Об'єктний аналіз предметної області і постановка завдання	05.05.2024	
2	Огляд літератури з предметної області	10.05.2024	
3	Обґрунтування структури додатку та інструментарію для розробки	15.05.2024	
4	Розробка структури програмного модулю, вибір даних для обробки та експериментів.	18.05.2024	
5	Тестування системи	22.05.2024	
6	Оформлення пояснювальної записки	30.05.2024	
7	Нормоконтроль пояснювальної записки, її перевірка антиплагіатною системою	01.06.2024	
8	Оформлення супутніх матеріалів (розробка графічного матеріалу, тощо) та рецензування роботи	12.06.2024	
9	Написання доповіді	12.06.2024	
10	Захист роботи	17.06.2024	

Студент



Погосян С.М.,

(прізвище та ініціали)

Керівник роботи

Башков Є.О.

(прізвище та ініціали)

АНОТАЦІЯ

Погосян Сурен Мелікович. Модуль підтримки зв'язку між співрозмовниками для анонімного чату / Випускна кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерне програмне забезпечення». – ДВНЗ ДонНТУ, Луцьк, 2024.

Об'єктом дослідження є процеси обміну інформації в соціальних мережах.

Предметом дослідження є методи організації взаємодії учасників соціальних мереж та чатах.

Метою роботи є створення мобільного чату з посиленою підтримкою анонімності користувачів.

Виходячи з мети, завданням даної дипломної роботи є аналіз сучасних анонімних чатів; розроблення програмної підтримки анонімного спілкування між користувачами.

Проведено порівняльний аналіз переваг та недоліків реальних анонімних чатів та їх різновидів.

Розроблено додаток, який включає серверну та клієнтську частини, що реалізовані використовуючи мову JavaScript.

Практичне значення роботи полягає в розробці програмної підтримки спілкування онлайн анонімно.

Дипломна робота включає, 3 таблиці, 36 рисунків, 19 джерела інформації, загалом 100 сторінок.

Ключові слова: МОБІЛЬНИЙ ДОДАТОК, REACT NATIVE, DATABASE, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС.

ABSTRACT

Suren Melikovych Pohosyan. Module for supporting communication between interlocutors for anonymous chat / Graduation qualification work for obtaining a bachelor's degree in specialty 121 "Engineering software". - DVNZ DonNTU, Lutsk, 2024.

The object of the study is the process of information exchange in social networks.

The subject of the study is methods of organizing the interaction of participants in social networks and chat rooms.

The goal of the work is to create a mobile chat with enhanced support for user anonymity.

Based on the goal, the task of this thesis is the analysis of modern anonymous chats; development of software support for anonymous communication between users.

A comparative analysis of the advantages and disadvantages of real anonymous chats and their varieties was carried out.

An application has been developed, which includes server and client parts implemented using the JavaScript language.

The practical significance of the work lies in the development of software support for anonymous online communication.

The thesis includes 3 tables, 36 figures, 19 sources of information, a total of 100 pages.

Keywords: MOBILE APPLICATION, REACT NATIVE, DATABASE, USER INTERFACE.

ЗМІСТ

ВСТУП.....	8
1 ОГЛЯД АНОНІМНИХ ЧАТІВ	9
1.1 Визначення звичайного чату.....	9
1.2 Особливості анонімних чатів у порівнянні зі звичайними чатами	12
1.3 Огляд та оцінка анонімних чатів	14
1.3.1 Чат Yik Yak [2].....	14
1.3.2 Чат Omegle [4].....	16
1.3.3 Чат Ask.fm [6].....	17
1.3.4 Чат Whisper [8]	18
1.3.5 Чат Chatous [9]	19
1.3.6 Чат Sarahah [11].....	22
1.3.7 Чат Kik [13]	23
1.3.8 Чат Slack [14].....	24
1.3.9 Чат ChatGig [16]	26
1.4 Висновки за розділом	27
2 ПРОЕКТУВАННЯ АРХІТЕКТРИ ПРОГРАМНОЇ СИСТЕМИ	28
2.1 Функціональні, нефункціональні та бізнес вимоги	28
2.2 Апаратні та програмні вимоги	28
2.3 Інструментарій.....	28
2.4 Побудова діаграми діяльності	30
2.5 Побудова діаграми послідовності	31
3 РОЗРОБКА КЛІЄНТСЬКОЇ ТА СЕРВЕРНОЇ ЧАСТИНИ.....	32

3.1 Структура проекту.....	32
3.2 Розробка клієнтського додатку.....	32
3.3 Розробка серверної частини.....	46
3.4 Розробка модуля підтримки зв'язку між співрозмовниками для анонімного чату.....	54
ВИСНОВОК	55
ПЕРЕЛІК ДЖЕРЕЛ	56
Додаток А	58
Зауваження нормоконтролера	58
Додаток Б	59
Фрагменти лістинги програми.....	59

ВСТУП

У сучасному світі все більше людей віддають перевагу знайомствам в Інтернеті перед зустріччю у реальному житті через швидкість, легкість та безпеку. Багато з нас відчувають постійний дефіцит часу, тому віртуальні платформи дозволяють нам швидко знайти співрозмовників, знайти спільні інтереси та розвинути відносини, не витрачаючи час на особисті зустрічі. Крім того, в Інтернеті ми можемо контролювати свою приватність та вибирати, з ким хочемо спілкуватися, що робить процес знайомства більш безпечним та комфортним.

Історія анонімних чатів нерозривно пов'язана з розвитком Інтернету та соціальних мереж. Вони почали з'являтися наприкінці 20-го століття, коли Інтернет тільки почав набирати популярність. Спочатку це були прості текстові чати, де користувачі могли обмінюватися повідомленнями без необхідності реєстрації чи надання особистої інформації.

Анонімні чати виникли як реакція на потребу користувачів інтернету у приватному та вільному спілкуванні без обмежень. Вони дозволяють людям обговорювати будь-які теми, виражати свої думки та почуття, не розкриваючи свою особисту інформацію. Це забезпечує атмосферу відкритості та дозволяє зустрічати нових людей з усього світу без обмежень географії чи соціального статусу.

1 ОГЛЯД АНОНІМНИХ ЧАТІВ

1.1 Визначення звичайного чату

Звичайний чат - це онлайн-платформа, де користувачі можуть вести обмін повідомленнями у реальному часі за допомогою тексту, аудіо або відео. У порівнянні з анонімними чатами, у звичайних чатах користувачі зазвичай реєструються, надаючи свої реальні дані, такі як ім'я, фотографія та електронна адреса.

Ключові характеристики звичайних чатів:

- реєстрація: для участі в чаті потрібна реєстрація з реальними даними;
- профілі: користувачі зазвичай мають профілі, де вони можуть представити себе в онлайн просторі;
- список друзів: користувачі можуть додавати один одного до друзів, щоб спілкуватися між собою;
- групи: у чатів часто є групи, присвячені певним темам чи інтересам;
- модерація: модератори стежать за дотриманням правил та видаляють неприйнятний контент у чатах.

Цілі використання звичайних чатів:

- спілкування: підтримка зв'язку з друзями, сім'єю та близькими;
- знайомства: пошук нових друзів, однодумців чи романтичних партнерів;
- обговорення тем: теми обговорень включають обмін ідеями, думками і інформацією на різні тематичні аспекти;
- розваги: участь у іграх, вікторинах та інших розважальних заходах;
- отримання підтримки: пошук допомоги або поради в інших людей;

Переваги звичайних чатів:

- простота використання: звичайні чати зазвичай прості у використанні;
- доступність: широкий вибір чатів доступний на різних платформах, включаючи веб-сайти, додатки для смартфонів;
- Соціальна взаємодія: це чудова можливість розширити коло спілкування, познайомитися з новими людьми і вдосконалювати соціальні вміння;
- підтримка може слугувати джерелом допомоги та підтримки для тих, хто переживає складні періоди.

Недоліки звичайних чатів:

- розповсюдження дезінформації: у чатах може поширюватися дезінформація та неправдиві відомості;
- ризик кібербулінгу: анонімність у деяких чатах може призвести до кібербулінгу та небезпеки інших користувачів;
- інформація без підтвердження: неможливо бути впевненим у достовірності інформації, яку ви отримуєте від інших користувачів, оскільки їхню особистість не завжди підтверджено;
- залежність: інтенсивне захоплення онлайн-спілкуванням може призвести до залежності від чатів та відриву від реального життя;

Важливо:

- будьте обережні при використанні звичайних чатів: уникайте розкриття особистих даних, які можна використовувати для вашої ідентифікації;
- виявляйте ввічливість та повагу: уникайте використання образливих висловлювань та переслідування інших користувачів;
- Якщо хтось зіткнеться з кібербулінгом, дезінформацією чи іншою неприйнятною поведінкою, важливо повідомити про це модераторів чату;
- Не забувайте про реальне життя: Не замінюйте реальне спілкування

на віртуальне;

Загалом, звичайні чати можуть бути корисним засобом для спілкування, знайомств, обговорення тем та отримання підтримки. Однак необхідно використовувати їх з обережністю та розуміти їх можливі негативні сторони.

Анонімний чат – це онлайн-платформа, де люди можуть спілкуватися один з одним без розкриття своєї особи. Користувачі зазвичай реєструються під псевдонімами або аватарами, що дозволяє їм вільно висловлювати свої думки та почуття без страху засудження чи переслідування [1].

Особливості анонімного чату:

- основна перевага полягає в анонімності: користувачі не зобов'язані розкривати свої особисті дані, такі як ім'я, фотографії, місцезнаходження тощо;
- свобода самовираження: анонімність дозволяє людям бути більш відкритими та чесними у своїх висловлюваннях, не боячись бути засудженими;
- різноманітність тем: в анонімних чатах можна обговорювати будь-які теми, які вас цікавлять без обмежень;
- анонімні чати можуть бути важливим джерелом підтримки та розуміння для людей, які перебувають у складних життєвих ситуаціях.

Переваги анонімного чату:

- анонімність сприяє подоланню соціальних перешкод, дозволяючи людям спілкуватися з тими, з ким вони не могли б вступити в розмову;
- дослідження своєї особистості: у захищеному середовищі анонімних чатів люди можуть досліджувати свою особистість та виражати аспекти себе, які зазвичай залишаються прихованими;
- здобуття нових знань та досвіду: спілкування з людьми з різних верств суспільства може розширити ваш кругозір і дати вам нові знання та досвід;

- зняття стресу: анонімний чат може стати місцем, де ви можете розслабитися та поділитися своїми проблемами з кимось, хто вас не засудить.

Недоліки анонімного чату:

- кібербулінг: на жаль, анонімність може також використовуватися для кібербулінгу та загрози інших користувачів;

- дезінформація: в анонімних чатах може поширюватися дезінформація та неправдиві відомості;

- непідтверджена інформація: ви не можете бути впевнені в достовірності інформації, яку отримуєте від інших користувачів, оскільки їх особи не підтверджено;

- ризик залежності: надмірне захоплення анонімними чатами може призвести до залежності від онлайн-спілкування та відходу від реального життя.

Важливо:

- використовуйте анонімні чати з обережністю: уникайте передачі особистих даних;

- виявляйте ввічливість та повагу: уникайте використання образ та переслідування інших учасників чату;

- повідомляйте про порушення: якщо зіткнетеся з кібербулінгом, дезінформацією або іншою неприйнятною поведінкою, повідомите про це модераторів чату;

- не забувайте про важливість реальних взаємодій: не порівнюйте віртуальне спілкування реального.

Загалом, анонімні чати можуть стати корисним засобом спілкування, вираження себе та отримання підтримки. Тим не менш, важливо використовувати їх з обережністю та усвідомлювати можливі ризики.

1.2 Особливості анонімних чатів у порівнянні зі звичайними чатами

Анонімні чати:

- підвищена конфіденційність: користувачі не зобов'язані розкривати

свої особисті дані, такі як ім'я, фотографії, місцезнаходження тощо, що дає їм більшу свободу вираження та знижує ризик переслідування;

- відкритість та чесність: анонімність може спонукати людей до більш відкритого та чесного спілкування, оскільки їм не потрібно боятися засудження чи соціальних обмежень;

- можливість знайомства з новими людьми з різних країн дозволяє розширити своє коло спілкування, долаючи географічні та соціальні кордони;

- підтримка: анонімні чати можуть стати місцем підтримки та розуміння для людей, які переживають не легкі часи;

Звичайні чати:

- менша приватність: у більшості звичайних чатів потрібне надання реальних даних при реєстрації, що може обмежувати свободу вираження користувачів через побоювання бути засудженими;

- схильність до більш стриманого спілкування: у зв'язку з недостатньою анонімністю люди можуть бути обережнішими у своїх висловлюваннях, що може стримувати їхню відвертість і глибину розмов;

- обмеження у виборі тем: у звичайних чатах обговорення може бути обмежене форматом чату, інтересами учасників або правилами спільноти;

- відсутність підтримки: звичайні чати не завжди призначені для підтримки людей, які переживають важкі часи;

Загалом, анонімні чати надають більш високу конфіденційність, відкритість та свободу вираження, що може сприяти більш цікавим, глибоким та підтримуючим розмовам. Тим не менш, звичайні чати можуть бути переважними в ситуаціях, коли необхідно розкрити особистість або коли ви вважаєте за краще спілкуватися з людьми зі свого оточення.

Крім вищесказаного, варто зазначити:

- рівень модерації в анонімних чатах зазвичай нижчий порівняно із звичайними чатами, що може збільшити ризик кібербулінгу, поширення дезінформації та іншої неприйнятної поведінки;

– вибір між анонімним та звичайним чатом залежить від ваших цілей. Якщо вам потрібна анонімна підтримка або хочете поділитися своїми переживаннями, то анонімний чат може бути кращим. Якщо ж ви шукаєте знайомства або хочете обговорити конкретну тему, звичайний чат може бути більш ефективним варіантом.

1.3 Огляд та оцінка анонімних чатів

Сьогодні анонімні чати стали неодмінною складовою онлайн-культури. Вони пропонують можливості для анонімного спілкування, які відрізняються за функціональністю та форматом:

– текстові чати є найбільш поширеним видом анонімних чатів. Тут користувачі можуть обмінюватися повідомленнями;

– деякі анонімні чати надають можливість користувачам спілкуватися один з одним за допомогою відео зв'язку, що дозволяє наблизити спілкування як у реальному житті;

популярні також анонімні форуми, де користувачі можуть обговорювати різноманітні теми або питання, залишаючи коментарі або пости;

1.3.1 Чат Yik Yak [2]

Yik Yak — це мобільний який спочатку з'явився у 2013 році і був перезапущений у 2021 році. Доступний для iOS та для Android, він дозволяє обговорювати події в радіусі 5 миль (8,0 км) від їхньої локації. Це схоже на інші додатки для анонімного обміну, такі як Nearby, але відрізняється тим, що головна акцентується на обміні інформацією з тими, хто знаходиться поблизу користувача. Інтерфейс програми (рисунок 1.1) приємний для користувача.

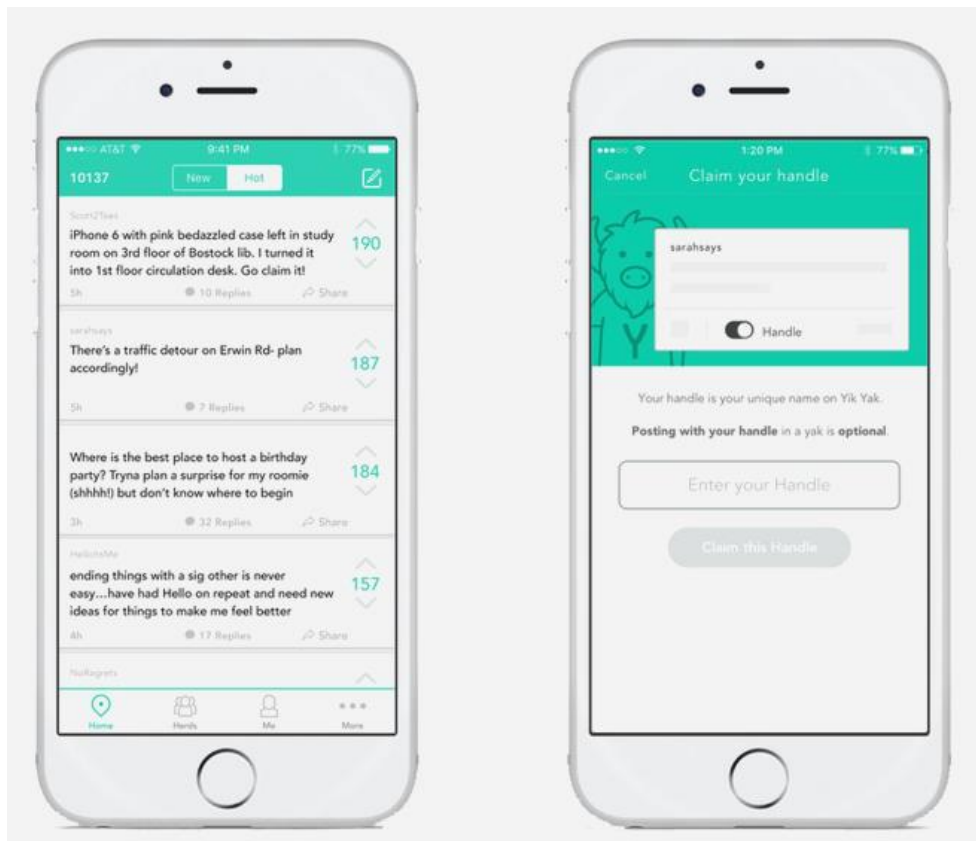


Рисунок 1.1 – Інтерфейс мобільного додатку. Джерело [3]

Переваги Yik Yak:

- анонімність дозволяє користувачам обговорювати теми, не розкриваючи свою особистість та не привертаючи зайву увагу до себе;
- цей додаток створює можливість обміну інформацією з іншими користувачами, що перебувають у тому ж регіоні;
- зручність полягає в тому, що інтерфейс додатка зручний у використанні, що призводить до його популярності;

Недоліки Yik Yak:

- зловживання може відбуватися через можливість анонімності, що може призвести до поширення образливого або неправдивого контенту без відповідальності;
- конфлікти можуть виникати через обговорення, що можуть спричинити конфлікти, оскільки учасники залишаються невідомими;

1.3.2 Чат Omegle [4]

Omegle – це веб сервіс, який дозволяє спілкуватися онлайн. Вибір співрозмовника відбувається абсолютно випадково, і спілкування проводиться повністю анонімно. Спілкування може відбуватися як у текстовому форматі, так і за допомогою веб камери. Інтерфейс програми продемонстровано на рисунку 1.2.

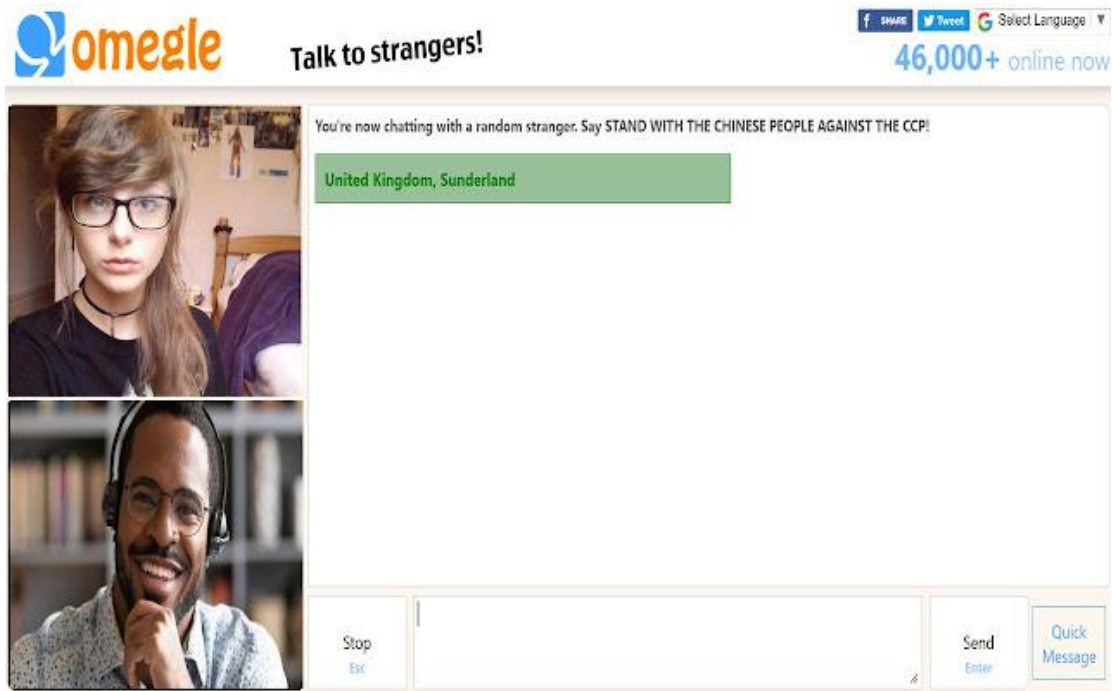


Рисунок 1.2 – Інтерфейс додатку. Джерело [5]

Переваги Omegle:

- однією з основних переваг Omegle є можливість спілкуватися з незнайомими людьми, що створює унікальну можливість для зацікавленості;
- користувачі мають можливість спілкуватися анонімно, також є можливість використання псевдонімів, що надає більшу свободу вираження своїх думок та почуттів без обмежень;
- в додатку є можливість спілкування як у текстовому форматі, так і за допомогою веб камери;

Недоліки Omegle:

- деякі додаткові функції та можливості Omegle доступні лише за

плату;

- відсутність контролю над контентом;

1.3.3 Чат Ask.fm [6]

Ask.fm - це онлайн-платформа, де користувачі можуть ставити один одному питання або відповідати на них анонімно. Це дозволяє виражати свої думки, ділитися досвідом і отримувати різноманітні відповіді на різні теми. Інтерфейс програми продемонстровано на рисунку 1.3.

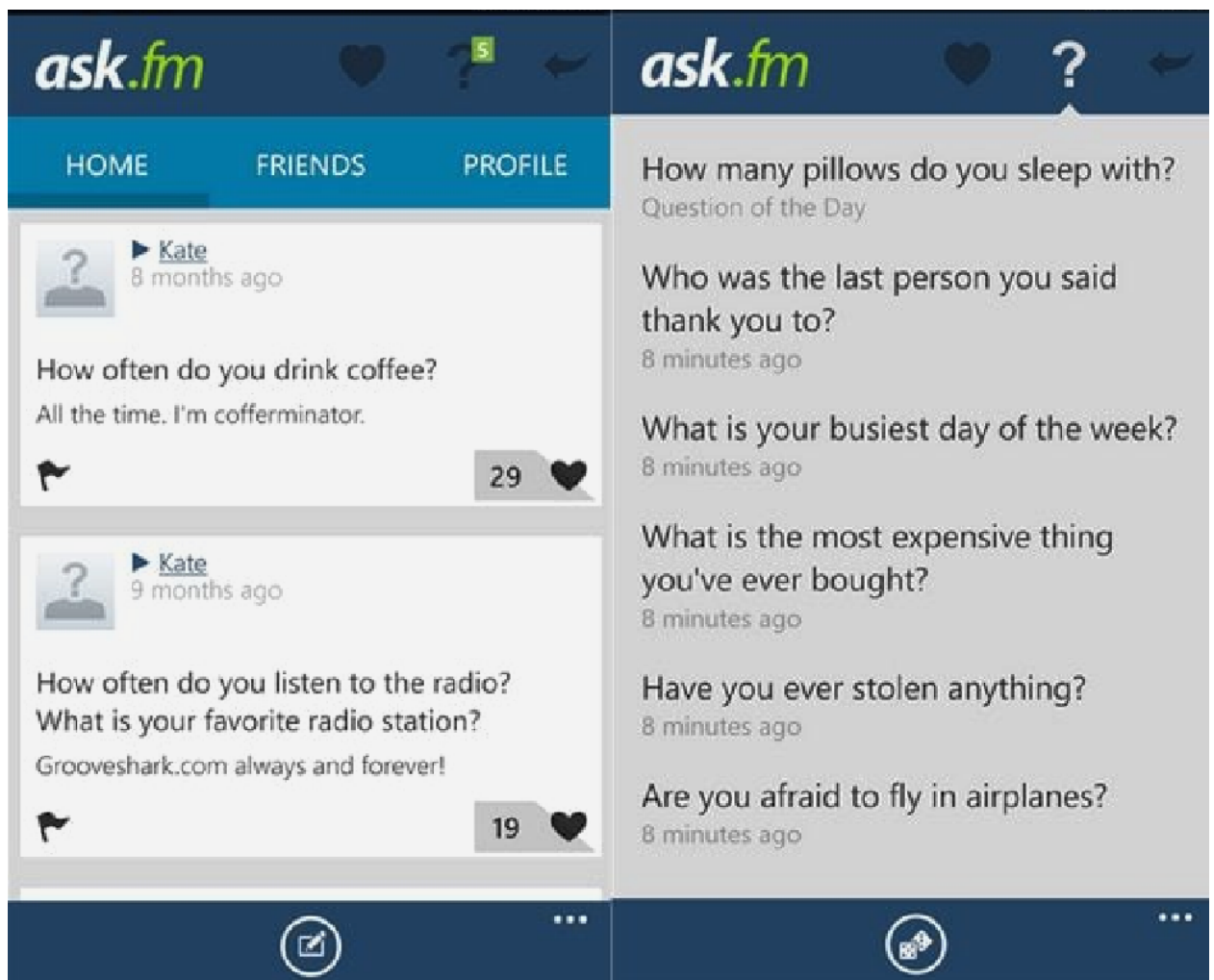


Рисунок 1.3 – Інтерфейс додатку. Джерело [7]

Переваги Ask.fm:

- анонімність на платформі Ask.fm дозволяє користувачам ставити питання та отримувати відповіді, не вказуючи свою особистість;

- різноманітність тем: платформа дозволяє обговорювати широкий спектр тем;
 - гнучкість формату: платформа пропонує різні формати взаємодії;
- Недоліки Ask.fm:
- ризик негативного контенту;
 - обмежена модерація;
 - втрата приватності;

1.3.4 Чат Whisper [8]

Whisper - це онлайн-спільнота, де мільйони людей з різних куточків світу можуть анонімно обмінюватися своїми думками, ідеями та останніми новинами, включаючи професійні обговорення. Інтерфейс програми продемонстровано на рисунку 1.4 – 1.5.

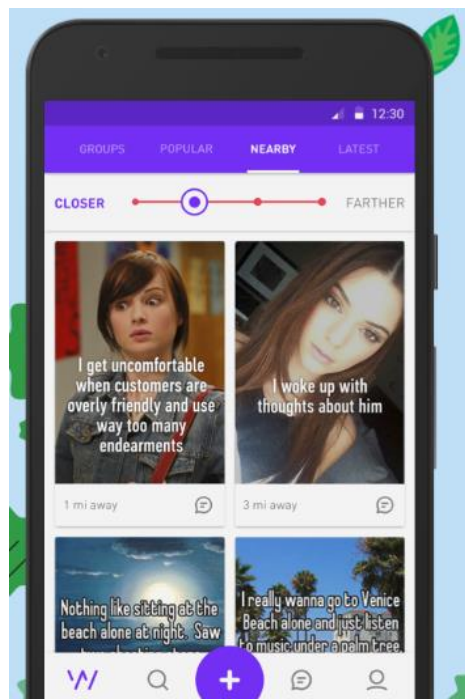


Рисунок 1.4 – Інтерфейс додатку. Джерело [8]

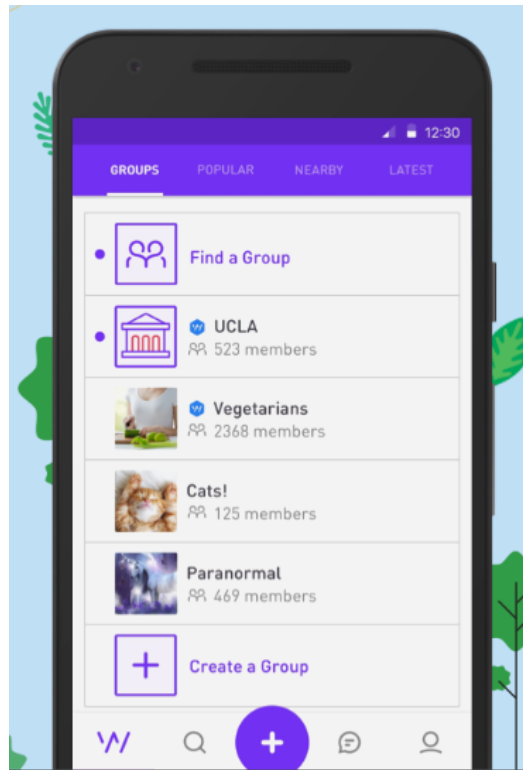


Рисунок 1.5 – Інтерфейс додатку. Джерело [8]

Переваги:

- анонімність на платформі Whisper створює можливість користувачам вільно виражати свої позиції та думки, що надає їм відчуття більшої свободи у виразі;
- комунікація без обмежень Whisper створює можливість обговорювати будь-які теми без обмежень;
- широкий аудиторіум мільйонами користувачів, що використовують цей додаток, швидко знаходити співрозмовника стає набагато простіше завдяки широкій аудиторії;

Недоліки:

- недостатня модерація контенту;
- обмежена конфіденційність;
- ризик негативних взаємодій;

1.3.5 Чат Chatous [9]

На платформі Chatous ви можете обговорювати теми, що вас цікавлять,

з людьми з усього світу. Це інколи призводить до непередбачуваних зустрічей.

Залишайтеся на зв'язку зі співрозмовниками, які вам цікавлять, та розширюйте свої спілкування за допомогою відеочату та обміну фотографіями. Інтерфейс програми продемонстровано на рисунку 1.6– 1.7.



Рисунок 1.6 – Інтерфейс додатку. Джерело [10]

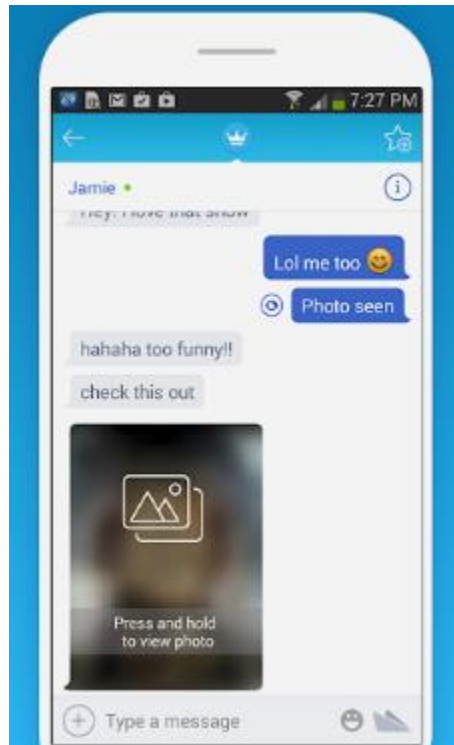


Рисунок 1.7 – Інтерфейс додатку. . Джерело [10]

Переваги:

- можливість вибору співрозмовників залежно від їхніх інтересів та залишати на зв'язку тільки тих, з ким вам приємно спілкуватися;
- різноманітність тем: немає обмеження щодо вибору теми розмови з співрозмовником;
- анонімність: ви можете спілкуватися анонімно, що дозволяє захистити вашу особисту інформацію;
- глобальний зв'язок: Chatous дозволяє спілкуватися з людьми з усього світу;

Недоліки:

- ризик небезпеки: на будь-якій соціальній платформі існує можливість натрапити на небажаних або потенційно небезпечних співрозмовників;
- залежність від інтернет-з'єднання;

1.3.6 Чат Sarahah [11]

Sarahah - це платформа, яка дозволяє користувачам анонімно пересилати повідомлення один одному. Додаток спочатку був розроблений як інструмент для надання конструктивного фідбеку, але з часом став популярним для анонімного спілкування загалом. Користувачі мають можливість створювати акаунти та отримувати анонімні повідомлення від інших відвідувачів. Інколи це може бути корисним для надання можливості людям виражати свої думки безпосередньо. Інтерфейс програми продемонстровано на рисунку 1.8.

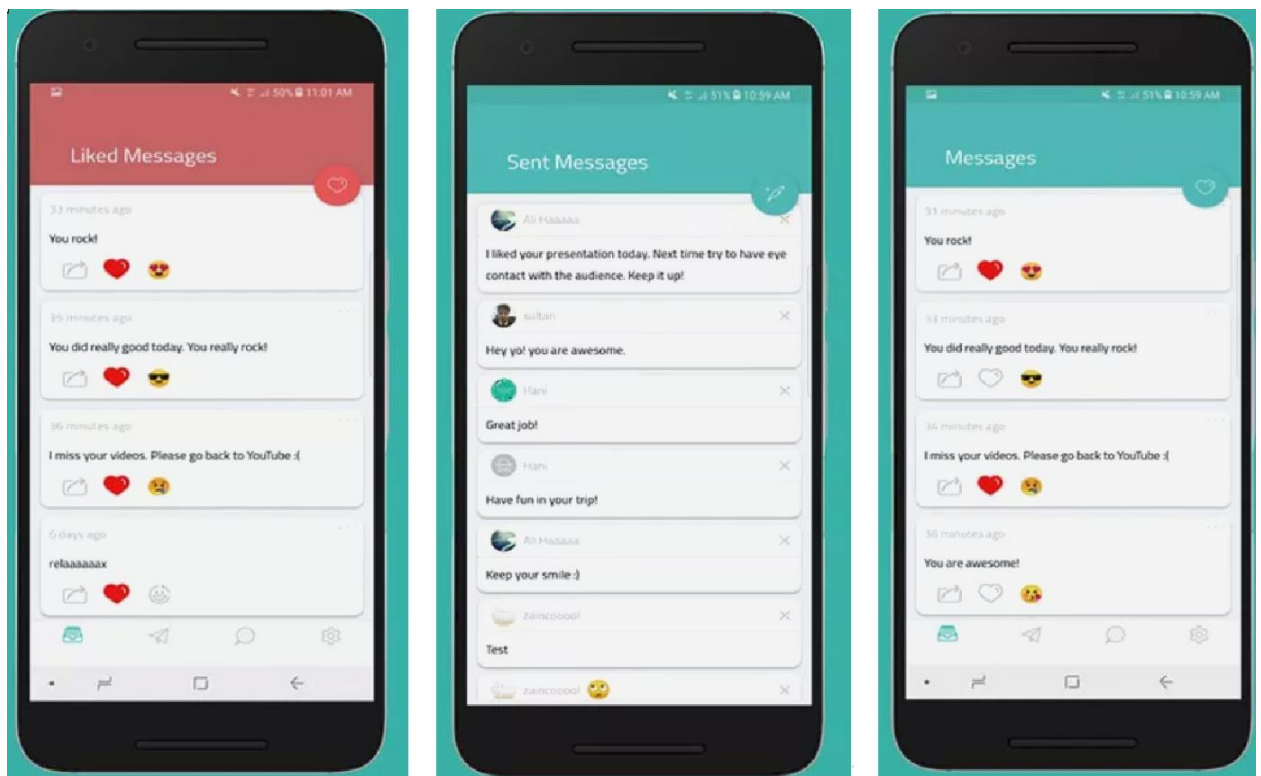


Рисунок 1.8 – Інтерфейс додатку. Джерело [12]

Переваги:

- можливість отримання фідбеку: початкова мета додатку була надавати можливість отримання корисних порад та вражень, сприяючи конструктивному обміну думками та зворотному зв'язку;
- простота використання: Sarahah має простий інтерфейс, що дозволяє користувачам легко орієнтуватися у додатку;

– анонімність: користувачі можуть відправляти повідомлення анонімно;

Недоліки:

- брак контролю над контентом: платформа може бути вразливою небажаного чи шкідливого вмісту через відсутність адекватної модерації;
- дуже багато реклами у додатку;
- інтерфейс програми не налаштований на різні мови;

1.3.7 Чат Kik [13]

Kik – це месенджер та соціальна платформа, яка дозволяє користувачам обмінюватися інформацією, такими як фотографії, відео та іншими мультимедійними файлами. Основна перевага Kik полягає в тому, що він не потребує введення особистої інформації для реєстрації. Крім того, Kik має функції для створення групових чатів, спільних розважальних ігор та інші можливості. Інтерфейс програми продемонстровано на рисунку 1.9– 1.10.

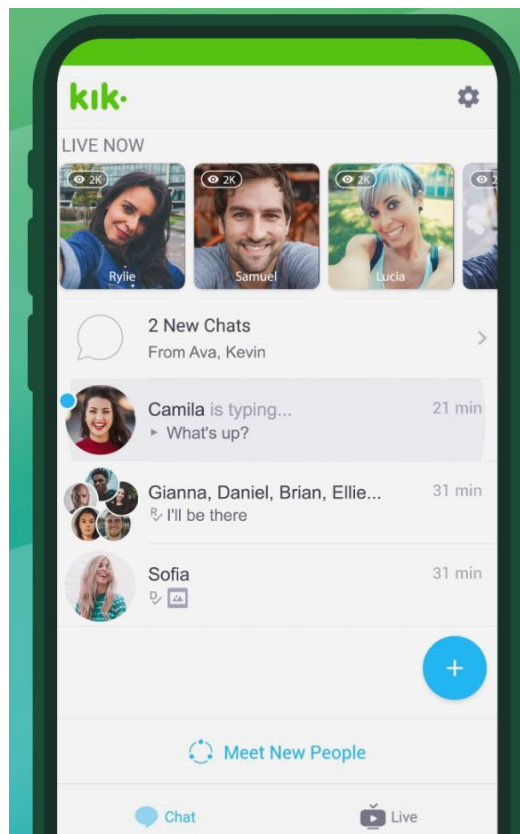


Рисунок 1.9 – Інтерфейс додатку. Джерело [13].

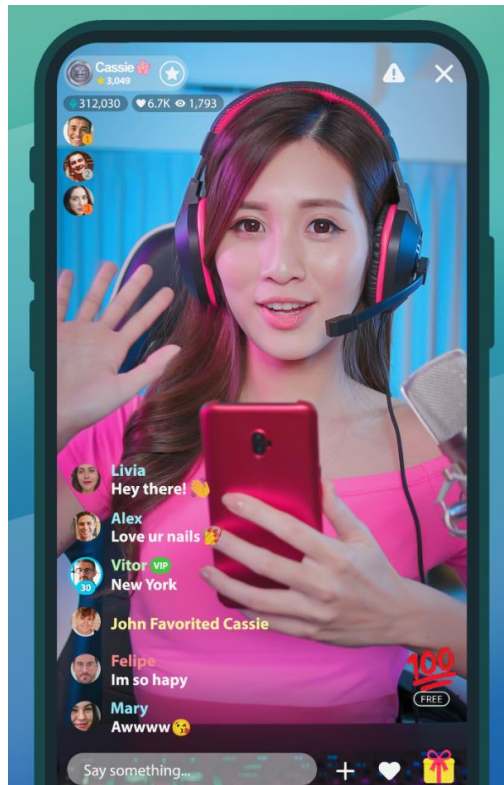


Рисунок 1.10 – Інтерфейс додатку. Джерело [13]

Переваги:

- відсутність потреби вводу особистої інформації;
- Kik пропонує багато можливостей для спілкування, такі як текстові повідомлення, стікери, чати та ігри;
- платформа має велику активну аудиторію, що дає можливість знайти співрозмовника дуже швидко;

Недоліки:

- комунікації на Kik не завжди шифруються, що може призвести до негативних наслідків;
- надмірне використання реклами яке заважає використанню додаток;
- існує ризик стикнутися зі спамом, шахрайством або небажаними контактами;

1.3.8 Чат Slack [14]

Slack надає широкий спектр функцій, схожих на IRC, включаючи створення тематичних каналів, приватних груп та можливість прямого обміну

повідомленнями. Весь контент, включаючи файли та розмови, доступний для пошуку. Користувачі можуть виражати свою реакцію за допомогою смайлів на будь-яке повідомлення. Історія повідомлень у Slack обмежена 10 000 останніми повідомленнями у безкоштовному плані. Інтерфейс програми продемонстровано на рисунку 1.11– 1.12.

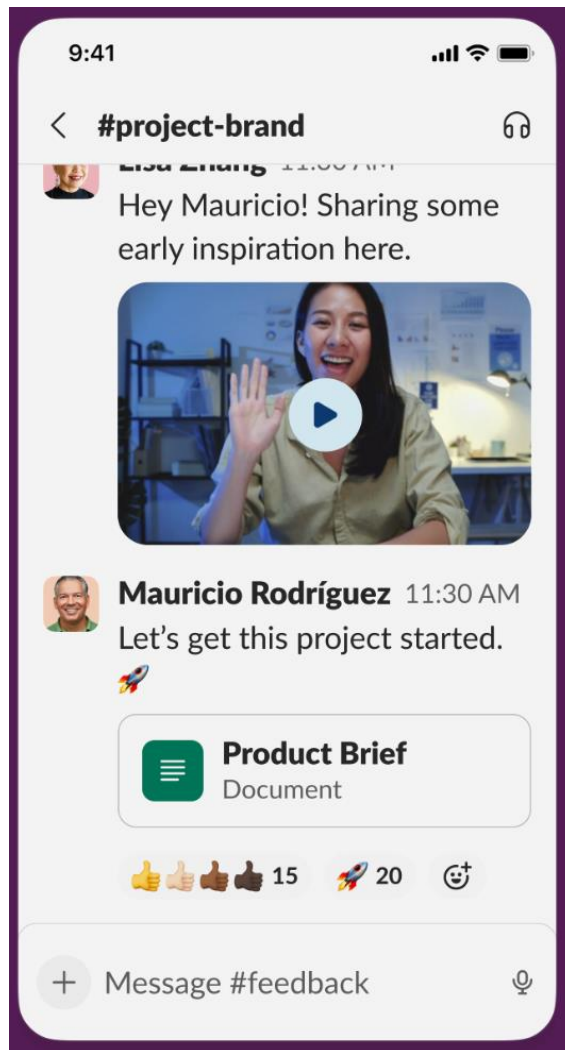


Рисунок 1.11 – Інтерфейс додатку. Джерело [15]

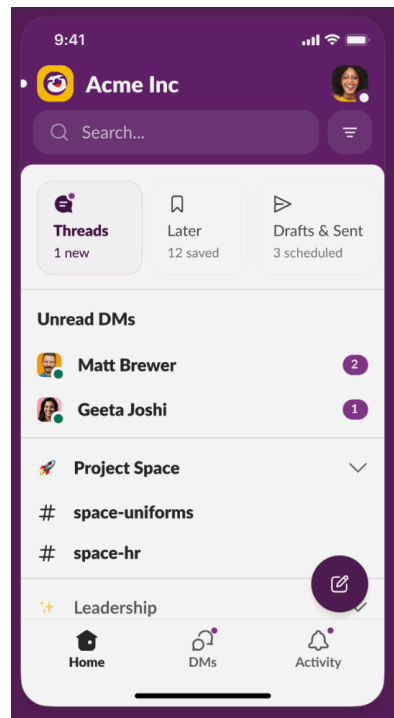


Рисунок 1.12 – Інтерфейс додатку. Джерело [15]

Переваги:

- повідомлення в Slack доставляються швидко, є багато функцій, таких як смайли та реакції, роблять спілкування зручним та ефективним;
- інтеграція з іншими сервісами;
- slack надає зручні інструменти для організації комунікації у вигляді тематичних каналів та приватних груп.

Недоліки:

- обмеження історії повідомлень: У безкоштовному плані Slack обмежує історію повідомлень лише 10 000 останніми повідомленнями;
- залежність від Інтернет-з'єднання: Slack потребує постійного з'єднання з Інтернетом;
- багато функції обмежені, для їх доступу потрібно платити;

1.3.9 Чат ChatGig [16]

ChatGig – це додаток, завдяки цій пропозиції люди можуть спілкуватися з незнайомими людьми з усього світу у випадкових анонімних чатах. Користувачі мають можливість обмінюватися повідомленнями без реєстрації,

додаток є повністю безкоштовним. Інтерфейс програми показано на рисунку 1.13.

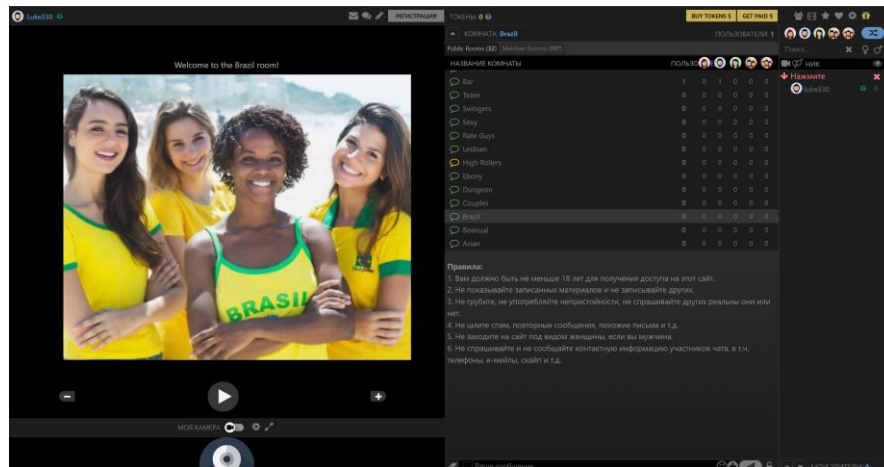


Рисунок 1.13 – Інтерфейс додатку. Джерело [16]

1.4 Висновки за розділом

Розглянуто основні анонімні чати і визначені переваги і недоліки кожного з них.

На сьогоднішній день існує безліч комплексних анонімних чатів для знаходження співрозмовника, і кожна з них охоплює велику кількість функцій для користувача. Більшість додатків мають підтримку відео чатів або текстових чатів. Спроба збільшити аудиторію, компанію кожним днем оновлюють та збільшують можливості користувачів для зручного спілкування в чаті. В більшості компанії які розробляють анонімні чати представляють користувачу вибір покупки преміуму або використання безкоштовної версії при покупці преміуму користувач отримує більше переваг, такі як використання стікерів або відео повідомлень. Також в деяких додатках використовують дуже багато реклами, які заважають користувачеві користуватися пропозицією і щоб відключити рекламу, користувач повинен купити преміум.

2 ПРОЕКТУВАННЯ АРХІТЕКТРИ ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні, нефункціональні та бізнес вимоги

Для створюваного продукту визначені наступні очікувані функції, вимоги до характеристик і бізнес-потреби:

- програма повинна підтримувати реєстрацію та авторизації для можливості входу в аккаунт;
- можливість запуску додатку в мобільних пристроях;
- програма повинна підтримувати можливість з'єднання з іншими користувачем;
- програма повинна підтримувати анонімність користувачів;
- можливість відправки повідомлення співрозмовнику;
- програма повинна підтримувати зміну співрозмовника;

2.2 Апаратні та програмні вимоги

Програмний додаток являє наступні вимоги:

- потрібна версія iOS 12.0 або новіший;
- розмір додатку 38,2 мб;

2.3 Інструментарій

У процесі розробки додатку використана мова програмування JavaScript, що широко використовується для створення різноманітних додатків.

Головні переваги JavaScript:

- простота і легкість освоєння: JavaScript порівняно простий для вивчення, особливо для програмістів-початківців. Його синтаксис інтуїтивно зрозумілий та логічний;
- виконання на стороні клієнта: JavaScript виконується у браузері користувача, що зменшує навантаження на сервер та покращує продуктивність веб-застосунків;

- інтерактивність та динамічність: JavaScript дозволяє створювати інтерактивні та динамічні елементи на веб-сторінках, такі як анімації, спливаючі вікна та форми з перевіркою даних у реальному часі;
- широка підтримка бібліотек та фреймворків: існує безліч бібліотек та фреймворків для JavaScript, таких як React, Angular, Vue.js та інші, які спрощують та прискорюють процес розробки;
- використання на серверній стороні: за допомогою Node.js JavaScript можна використовувати і на серверній стороні, що дозволяє створювати повнофункціональні серверні програми;

Також у процесі розробки було використано Node.js, це середовище виконання JavaScript, яке дозволяє запускати JavaScript-код на сервері.

Ключові переваги використання Node.js:

- єдина мова для фронтенду та бекенду: використання JavaScript як на клієнтській, так і на серверній стороні;
- спільнота та підтримка: Node.js має велику спільноту розробників, що забезпечує наявність великої кількості ресурсів, документації та підтримки;
- кросплатформенність: Node.js може працювати на різних платформах, включаючи Windows, Linux та macOS, що робить його універсальним інструментом для розробки серверних програм;
- масштабованість: Node.js легко масштабується як по вертикалі, так і по горизонталі, що робить його хорошим вибором для створення додатків, що масштабуються;

Недоліки використання Node.js:

- уразливості в сторонніх модулях: Node.js залежить від сторонніх модулів з npm(Node package manager), і деякі з них можуть містити вразливості. Управління безпекою залежностей може бути складним і потребує регулярних перевірок та оновлень.

2.4 Побудова діаграми діяльності

Для проєктованого анонімного додатку було побудовано діаграму діяльності, що відображає поведінку реєстрації та авторизації користувача.

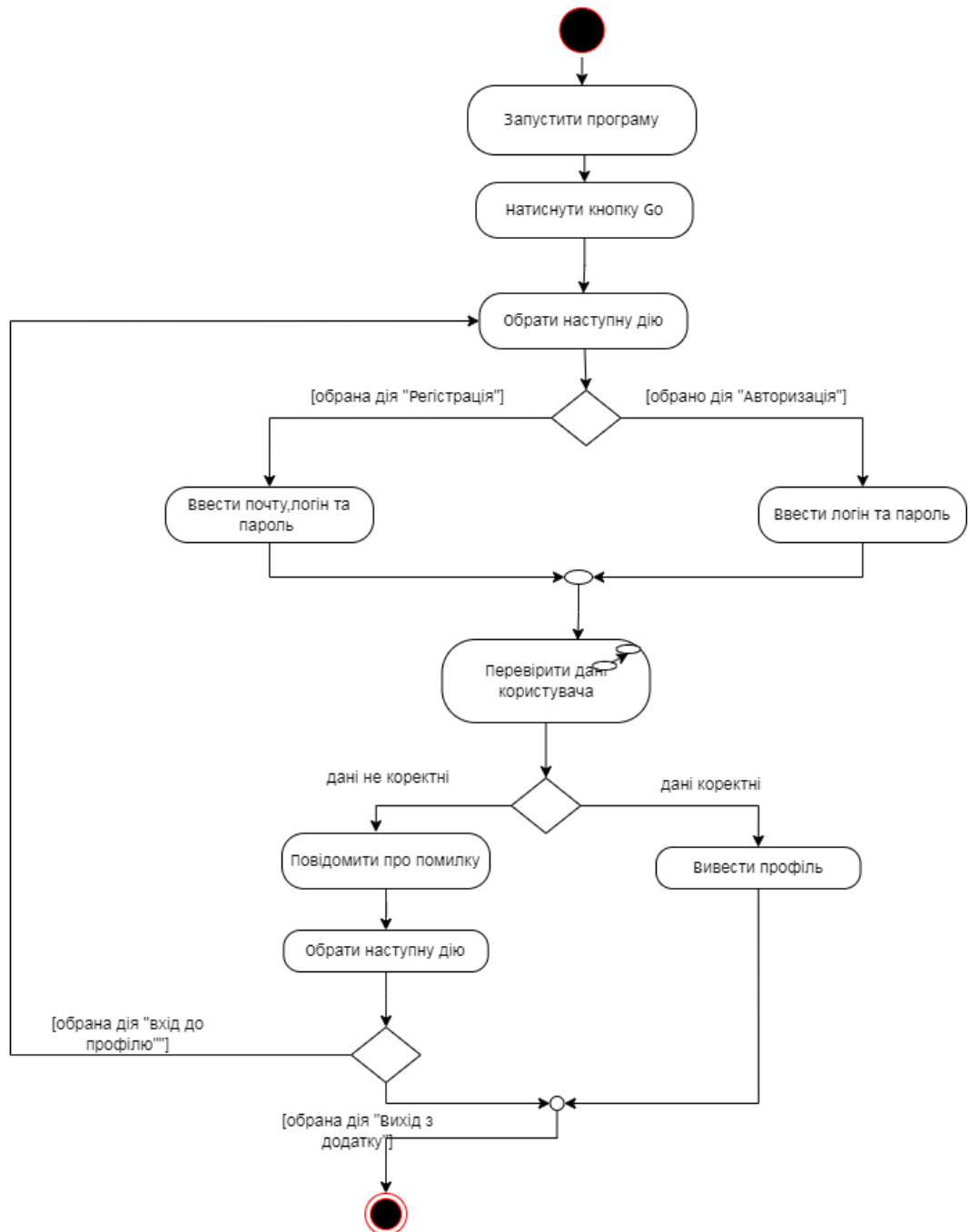


Рисунок 2.1 –Діаграма діяльності, що відображає авторизацію та реєстрацію користувача

Для потрапляння в початковий стан діаграми потрібно завантажити та запустити додаток.

2.5 Побудова діаграми послідовності

Для проєктованого додатку була виконана побудова діаграми послідовності, що відображає поведінку системи, характерну їй під час знаходження чату. Для потрапляння в місце, з якого починається діаграма послідовності, потрібно: зайти в свій профіль.

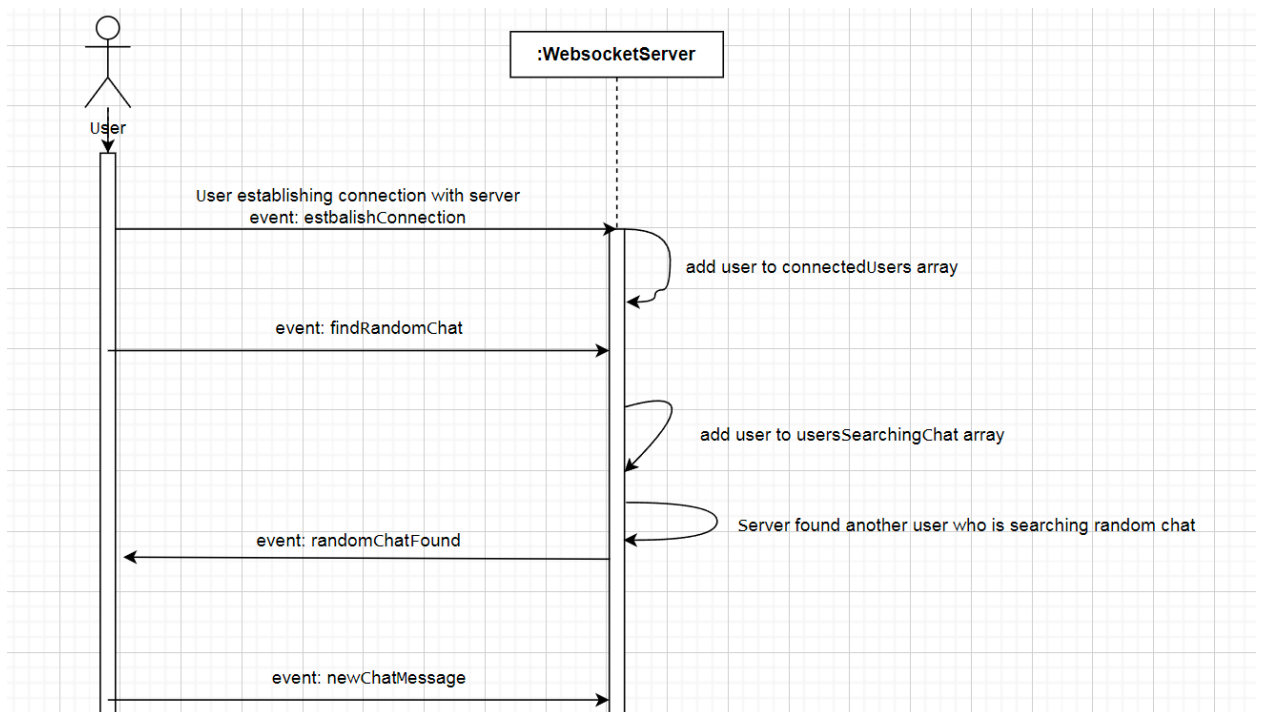


Рисунок 2.2 – Діаграма послідовності, що відображає алгоритм знаходження анонімного чату

3 РОЗРОБКА КЛІЄНТСЬКОЇ ТА СЕРВЕРНОЇ ЧАСТИНИ

3.1 Структура проекту

Платформа складається з трьох основних модулів:

- серверний додаток;
- клієнтський додаток;
- серверний додаток для обробки чатів;

3.2 Розробка клієнтського додатку

Для створення фронтенд частини програми був використаний фреймворк React Native. Він дозволяє розробнику створювати компоненти. Цей підхід є компонентним. Перевага такого підходу є те, що компоненти можна використовувати багаторазово, експортуючи компонент в різні частини програми. Компонентний підхід показано на рисунку 3.1.

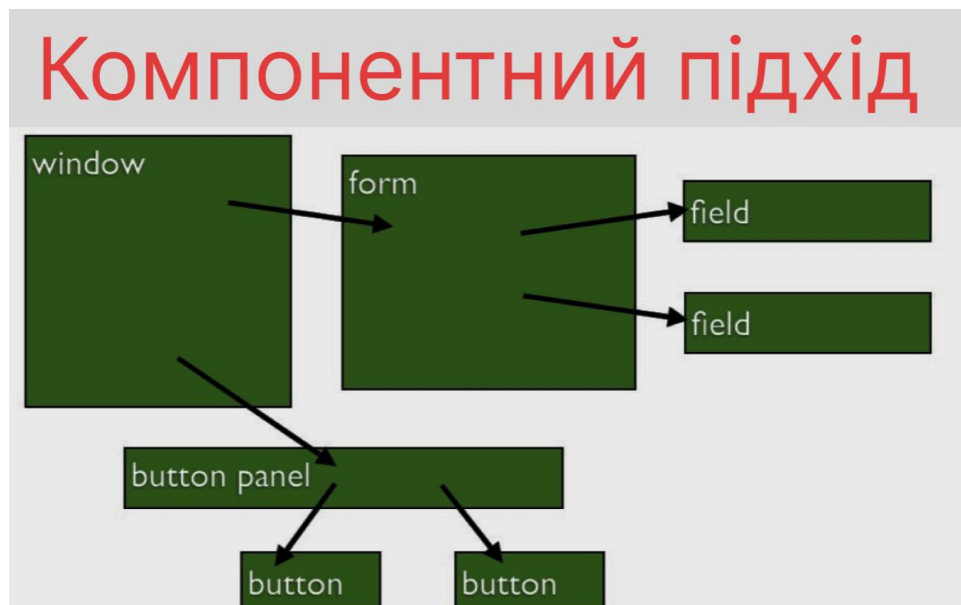


Рисунок 3.1 – Компонентний підхід

Для спрощення створення адаптивного додатку, застосовується Flexbox для інтерфейсів. Стили компонентів вносять через JavaScript, що дозволяють динамічно змінювати стилі залежно стану програми.

React Native надає бібліотеку React Navigation, він включає потужні інструменти для створення та управління навігацією в додатку, підтримуючи стекову, табличну та іншу навігацію.

Для розробки нативної програми було використано фреймворк React Native. Цей фреймворк користується популярністю серед програмістів, так він дозволяє використовувати JavaScript і React для розробки кросплатформових додатків, які можуть працювати як на iOS, так і на Android. Ось основні переваги та недоліки React Native.

Переваги React Native:

- єдиний код для iOS та Android: React Native дозволяє писати один код, який буде підтримуватися на обох платформах, що значно скорочує час та витрати на розробку;
- використання JavaScript: оскільки React Native є фреймворком JavaScript, розробники можуть використовувати безліч існуючих JavaScript-бібліотек та інструментів.
- розробники бачать зміни миттєво у розробці програми, без необхідності перезапуску програми, оскільки React Native має функцію hot reloading;
- доступ до нативних API: React Native дозволяє використовувати нативні модулі для доступу до функціональності апарату, такої як камера, GPS та інші;

Недоліки React Native:

- продуктивність: складні обчислення: React Native може поступатися нативним програмам у продуктивності, особливо при виконанні складних анімацій або інтенсивних обчислень;
- якість бібліотек: Не всі сторонні бібліотеки React Native мають активну підтримку, що може викликати проблеми при розробці;
- більший розмір: програми React Native можуть бути більшими за нативні програми;

- помилки та налагодження: налагодження та усунення помилок у React Native можуть бути складнішими. особливо якщо помилки виникають на рівні нативного коду.

React Native –потужний інструмент для розробки мобільних додатків, що пропонує безліч переваг, таких як єдиний код для iOS та Android. Однак, як і будь-яка технологія, вона має свої недоліки та переваги, які слід враховувати при виборі інструменту для конкретного проекту.

Структура проекту клієнтського додатка показано рисунку 3.2.

- `componenets` – директорія для зберігання компонентів;
- `utils` – містить функції JavaScript, які використовуються в компонентах;
- `ui (user interface)` – для зберігання користувацького інтерфейсу;
- `assests` – для зберігання картинок та іконок;
- `views` – сторінки програми;
- `node-modules` – містить всі залежності, необхідні для роботи проекту.

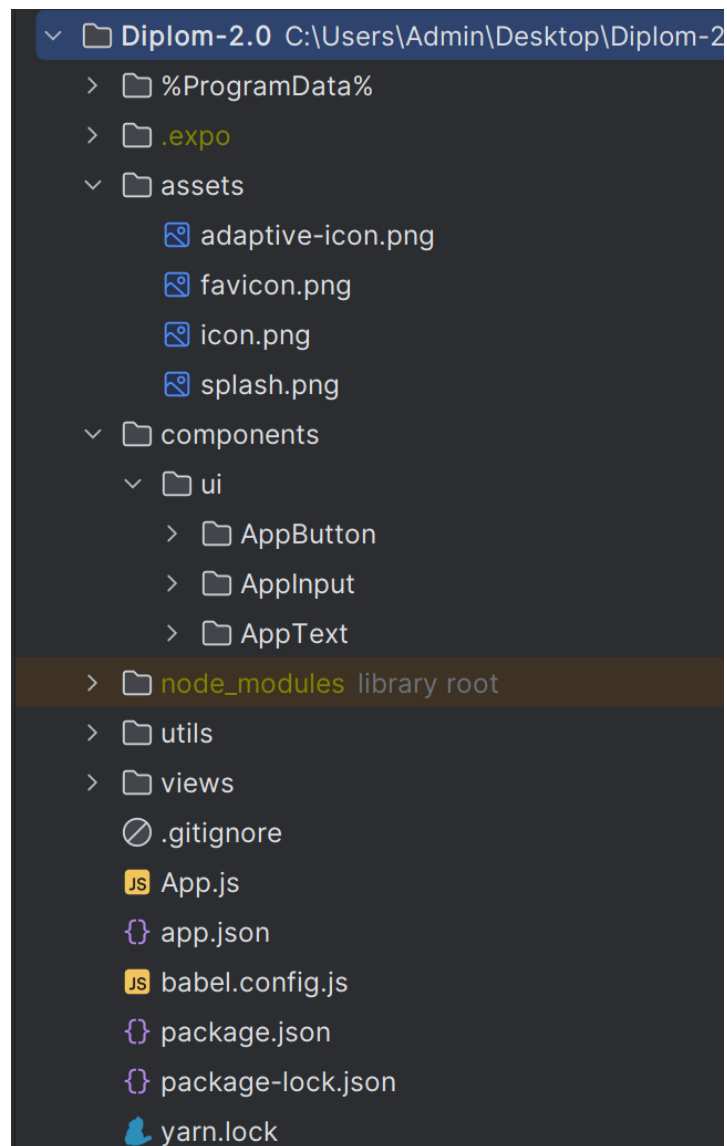


Рисунок 3.2 – Структура проекту клієнтського додатка

Для збірки файлів вихідного коду, статичних файлів, ресурсів клієнтської програми використовується Metro. Metro - це збирач JavaScript.

Він приймає початковий файл та повертає один файл JavaScript, який включає весь ваш код та його залежності. На рисунку 3.3 проілюстровано схему роботи Metro Bundler.

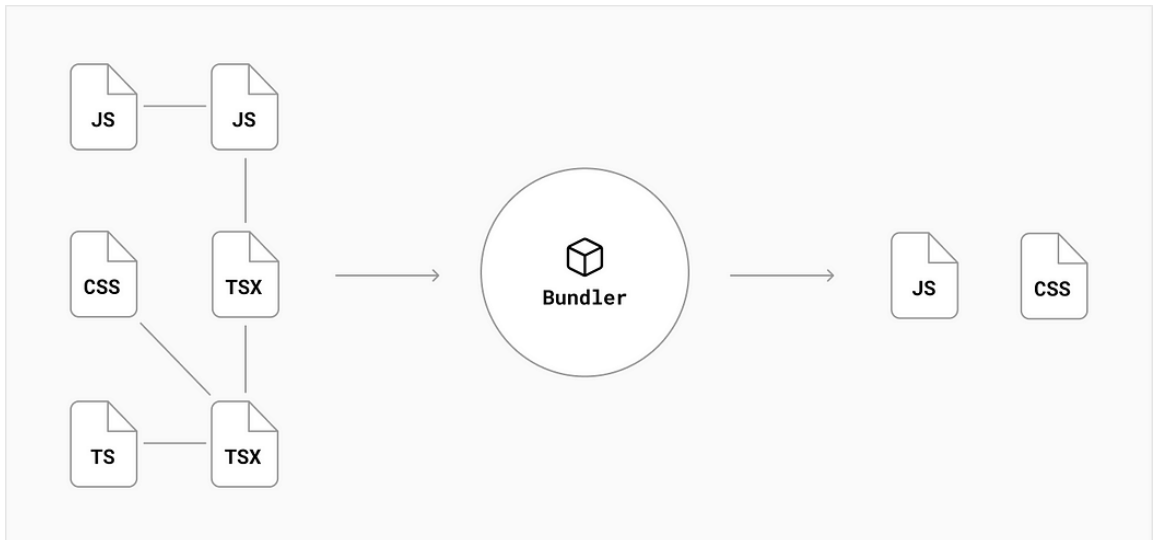


Рисунок 3.3 – Схема роботи Metro Bundler [18]

На рисунку 3.4, продемонстровано авторизацію користувача.

< Let's chat Log in 5G

LOGIN

Username

Password

LOGIN

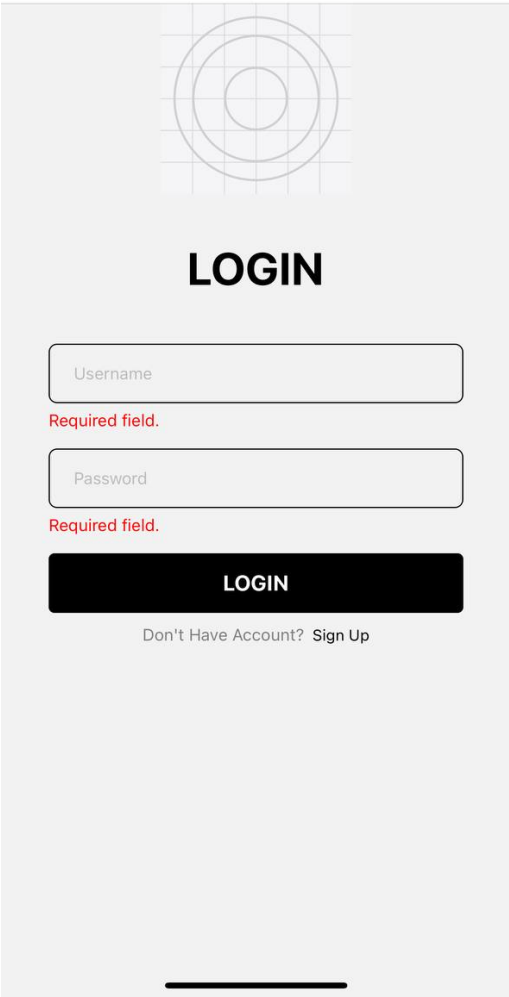
Don't Have Account? Sign Up

Рисунок 3.4 – Авторизація

На вікні авторизації є чотири форми:

- ім'я користувача;
- пароль;
- кнопка входу до профілю користувача;
- перехід на сторінку реєстрації.

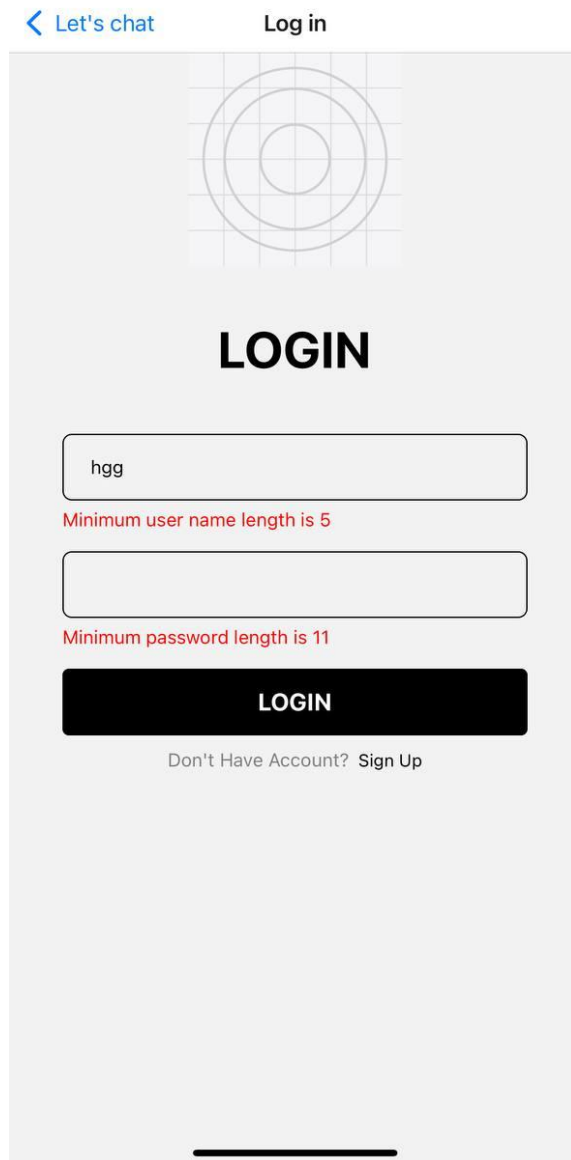
На рисунку 3.5 продемонстровано, якщо користувач не заповнить форму.



The screenshot shows a mobile application interface for a login screen. At the top, there is a navigation bar with a blue back arrow and the text "Let's chat" on the left, and "Log in" on the right. Below the navigation bar is a large, light gray rectangular area. Inside this area, at the top, is a circular profile picture placeholder with a grid pattern. Below the placeholder is the word "LOGIN" in bold, black, uppercase letters. Underneath "LOGIN" are two input fields. The first field is labeled "Username" and has a red error message "Required field." below it. The second field is labeled "Password" and also has a red error message "Required field." below it. Below the input fields is a black button with the word "LOGIN" in white, uppercase letters. At the bottom of the form area, there is a link that says "Don't Have Account? Sign Up".

Рисунок 3.5 – Перевірка на ведення даних

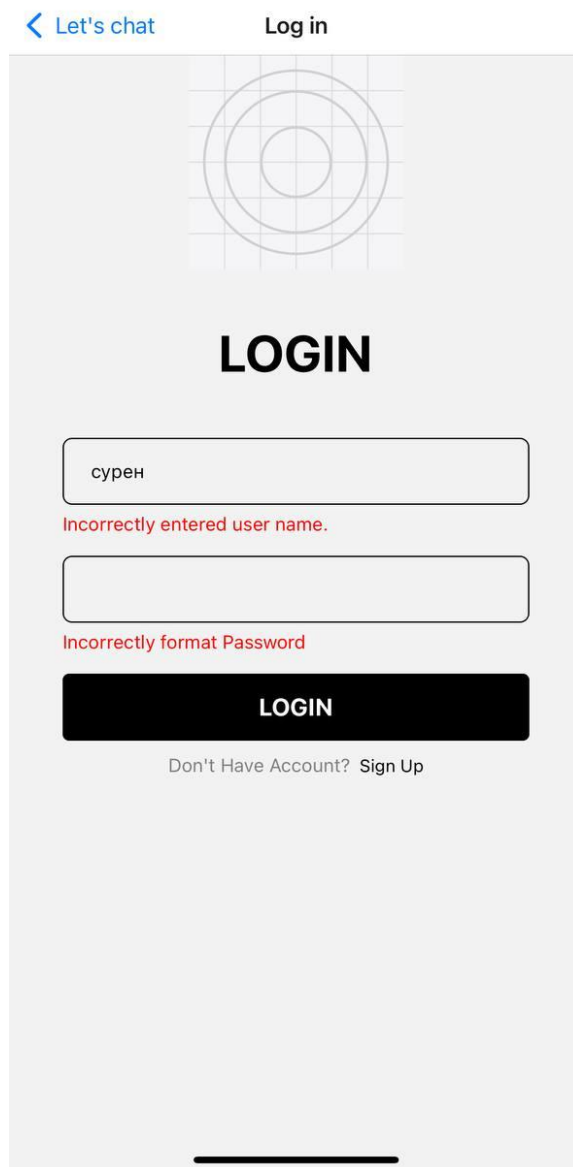
На рисунку 3.6 продемонстровано, якщо користувач не введе достатньо символів у певну форму.



The image shows a mobile app login screen. At the top, there are two links: a blue arrow pointing left followed by "Let's chat", and "Log in". Below these is a large, faint circular logo with a grid pattern. The word "LOGIN" is centered in bold black text. There are two input fields. The first field contains the text "hgg" and has a red error message below it: "Minimum user name length is 5". The second field is empty and has a red error message below it: "Minimum password length is 11". Below the fields is a black button with the word "LOGIN" in white. At the bottom, there is a link: "Don't Have Account? Sign Up".

Рисунок 3.6 – Перевірка на кількість символів

На рисунку 3.7 продемонстровано, якщо користувач введе кирилицю.



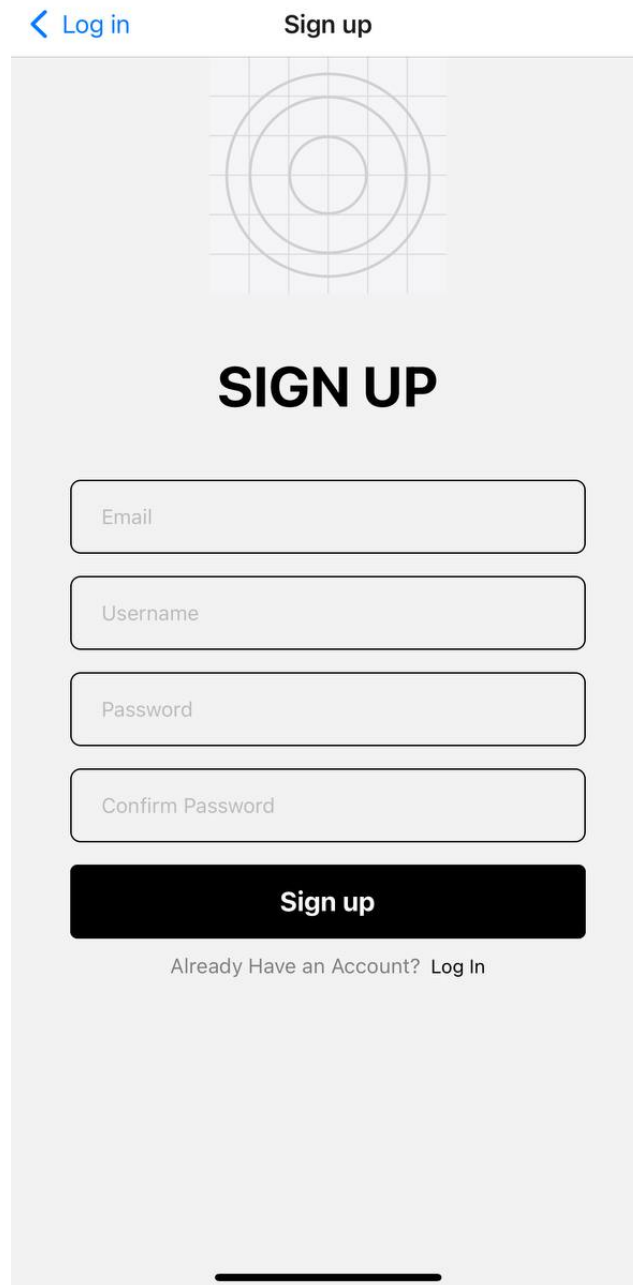
The image shows a mobile application login screen. At the top, there is a navigation bar with a blue back arrow and the text "Let's chat" on the left, and "Log in" on the right. Below the navigation bar is a large, light gray square containing a circular grid pattern, resembling a fingerprint scanner. The word "LOGIN" is displayed in bold, black, uppercase letters. Below this, there are two input fields. The first field contains the text "сурен" and has a red error message "Incorrectly entered user name." below it. The second field is empty and has a red error message "Incorrectly format Password" below it. Below the input fields is a black button with the word "LOGIN" in white, uppercase letters. At the bottom of the screen, there is a link that says "Don't Have Account? Sign Up".

Рисунок 3.7 – Перевірка логіна на кирилицю

На вікні реєстрації є шість форм:

- пошта;
- ім'я користувача;
- пароль;
- підтвердження паролю;
- кнопка входу до профілю користувача;
- перехід на сторінку авторизації.

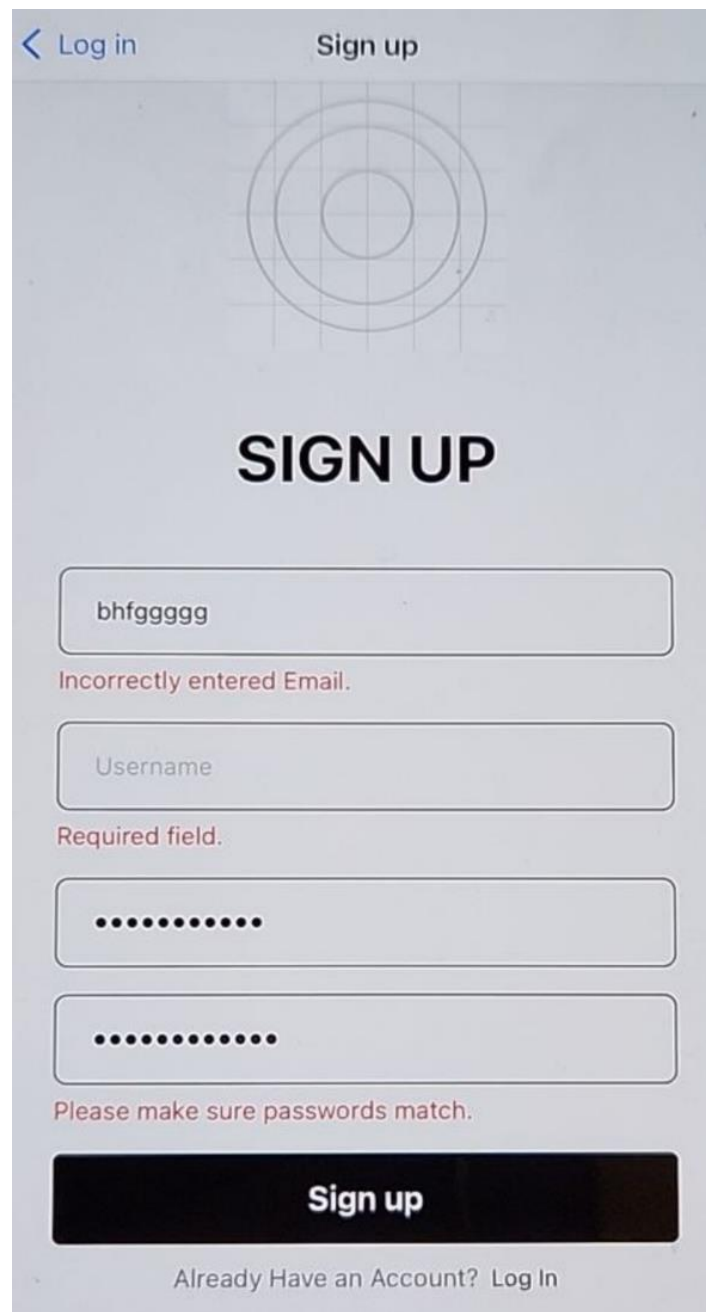
На рисунку 3.8 продемонстровано реєстрацію користувача.



The image shows a mobile application interface for user registration. At the top, there is a navigation bar with a blue back arrow and the text "Log in" in blue, and the text "Sign up" in black. Below the navigation bar is a large, light gray rectangular area containing the registration form. At the top of this area is a circular logo consisting of three concentric circles. Below the logo, the text "SIGN UP" is displayed in large, bold, black capital letters. Underneath the title, there are four input fields stacked vertically, each with a light gray border and a light gray background. The first field is labeled "Email", the second "Username", the third "Password", and the fourth "Confirm Password". Below the input fields is a black button with the text "Sign up" in white. At the bottom of the form area, there is a link that says "Already Have an Account? Log In" in a smaller, gray font. The entire form is centered on the screen.

Рисунок 3.8 – Реєстрація користувача

Перевірка на введення правильного формату пошти, пароля та підтвердження на рисунку 3.9.



The screenshot shows a mobile application interface for signing up. At the top, there is a navigation bar with a back arrow and the text 'Log in' on the left, and 'Sign up' on the right. Below the navigation bar is a large, faint circular graphic. The main heading 'SIGN UP' is centered in bold black text. Below the heading are four input fields. The first field contains the text 'bhfggggg' and has a red error message 'Incorrectly entered Email.' below it. The second field contains the placeholder text 'Username' and has a red error message 'Required field.' below it. The third and fourth fields are password fields, each containing ten dots. The fourth field has a red error message 'Please make sure passwords match.' below it. At the bottom of the form is a large black button with the text 'Sign up' in white. Below the button is a link that says 'Already Have an Account? Log In'.

Рисунок 3.9 – Перевірка правильності введення електронної пошти

При успішній реєстрації або авторизації користувача входить до його профілю. Це продемонстровано на рисунку 3.10.

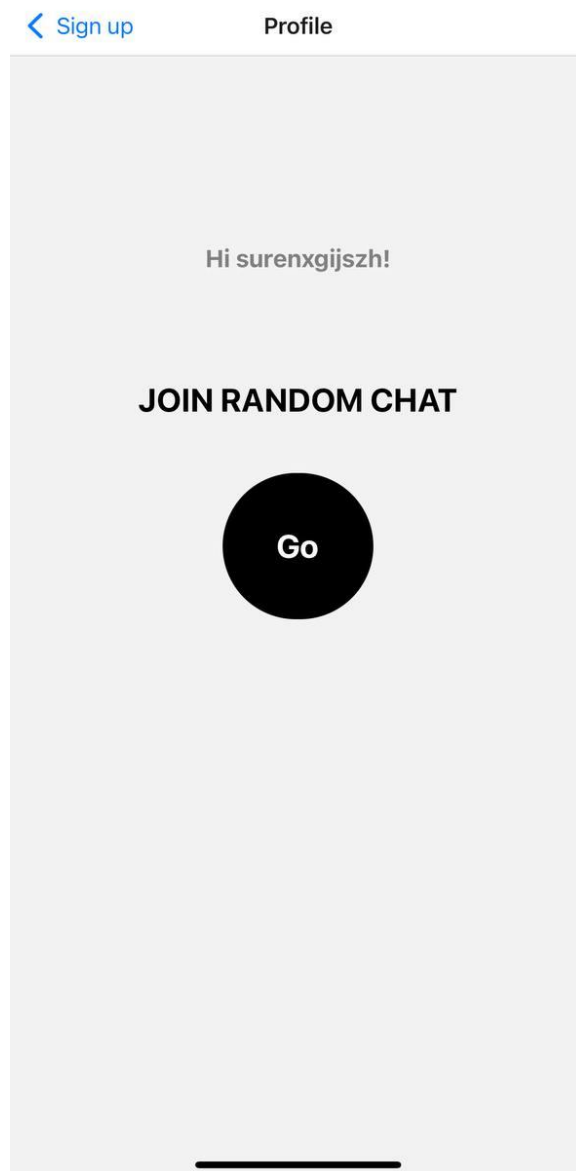


Рисунок 3.10 – Вхід до профілю користувача

При натисканні на кнопку Go перекидає на сторінку пошук співрозмовника. . Це продемонстровано на рисунку 3.11.

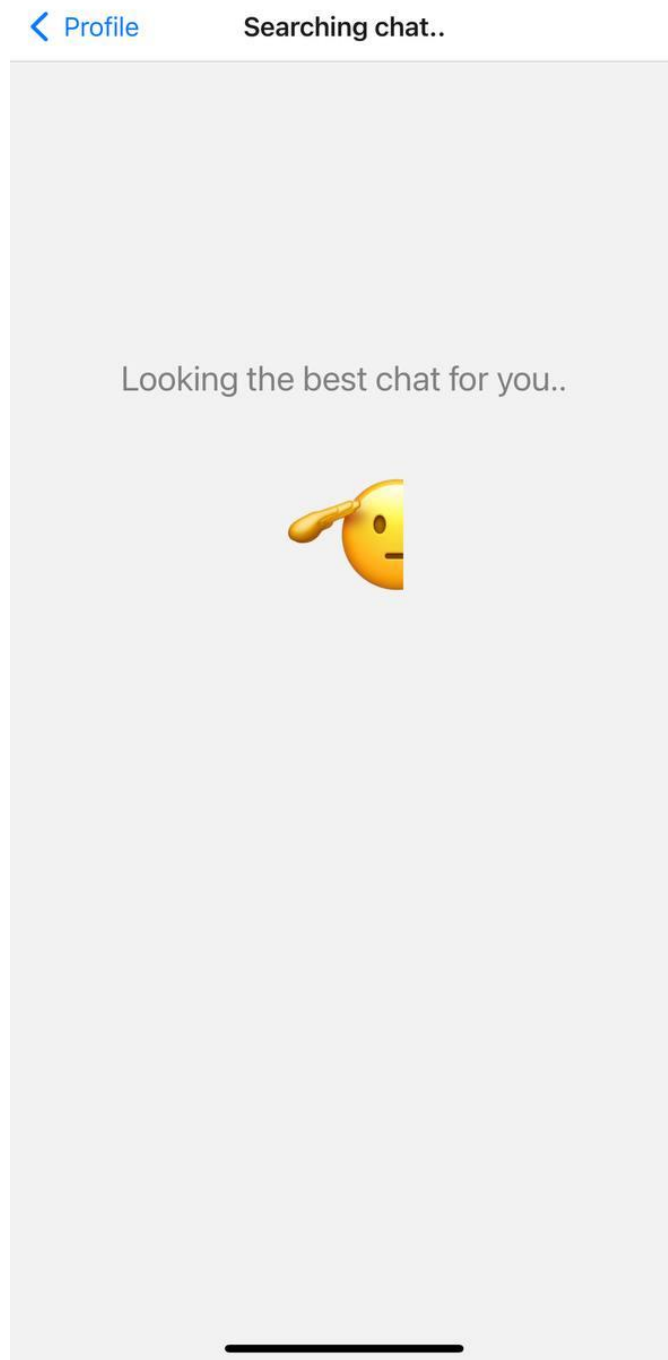


Рисунок 3.11 – Пошук користувача

Після знаходження співрозмовника, програма перенаправляє користувачів до кімнати та повідомляє їх про підключення співрозмовника. Це продемонстровано на рисунку 3.12.

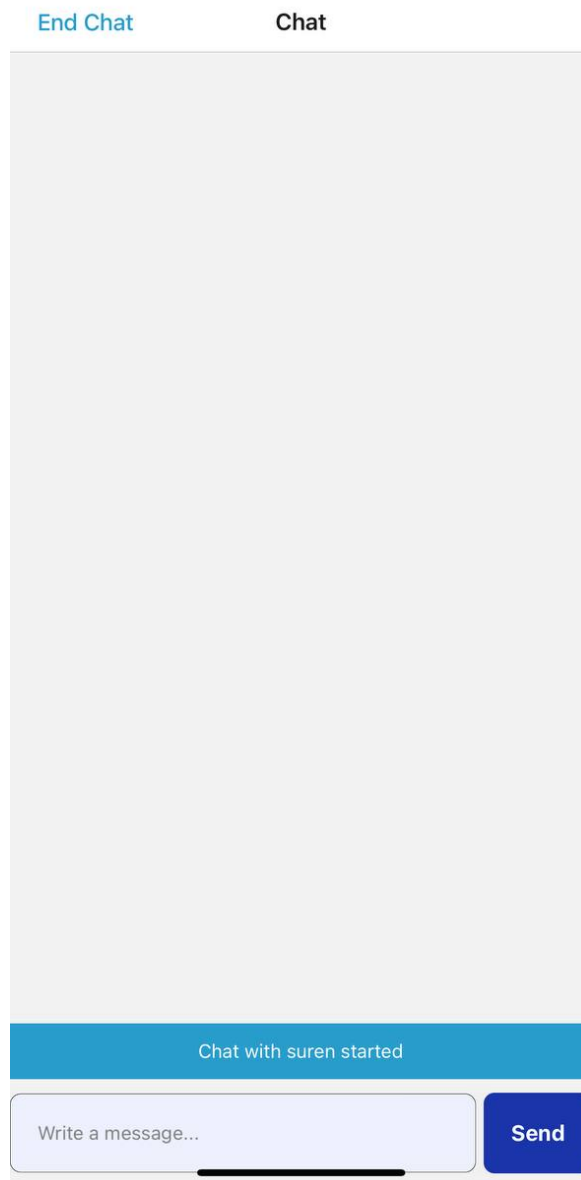


Рисунок 3.12 – Створена кімната для спілкування

Під час надсилання повідомлення відображається відправник, саме повідомлення та час надсилання. Коли користувач отримує повідомлення від іншого користувача, воно буде чорного кольору. Якщо ж користувач вирішить надіслати повідомлення, то колір його повідомлення буде синього. Це продемонстровано на рисунку 3.13.

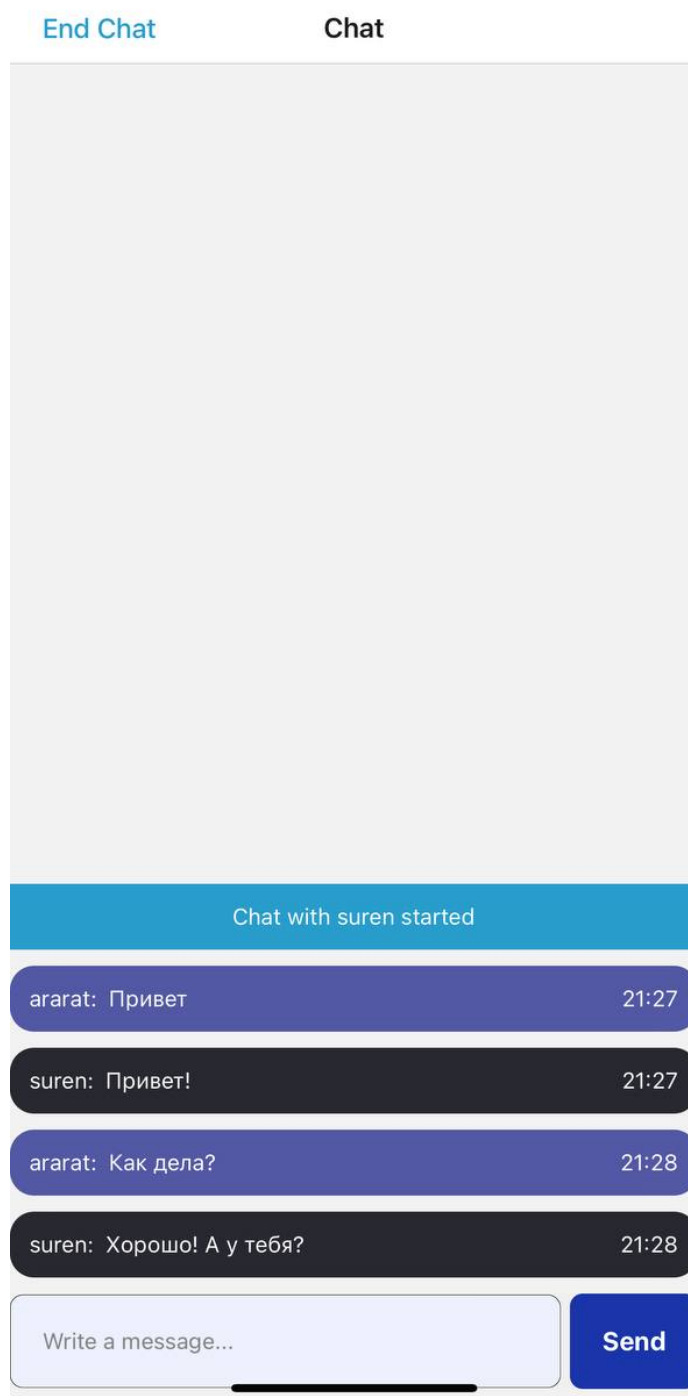


Рисунок 3.13 – Створена кімната для спілкування

Щоб вийти з чату, потрібно натиснути на кнопку End. При натисканні на цю кнопку всі учасники будуть повідомлені та перекинуті на свій профіль. Це продемонстровано на рисунку 3.14.

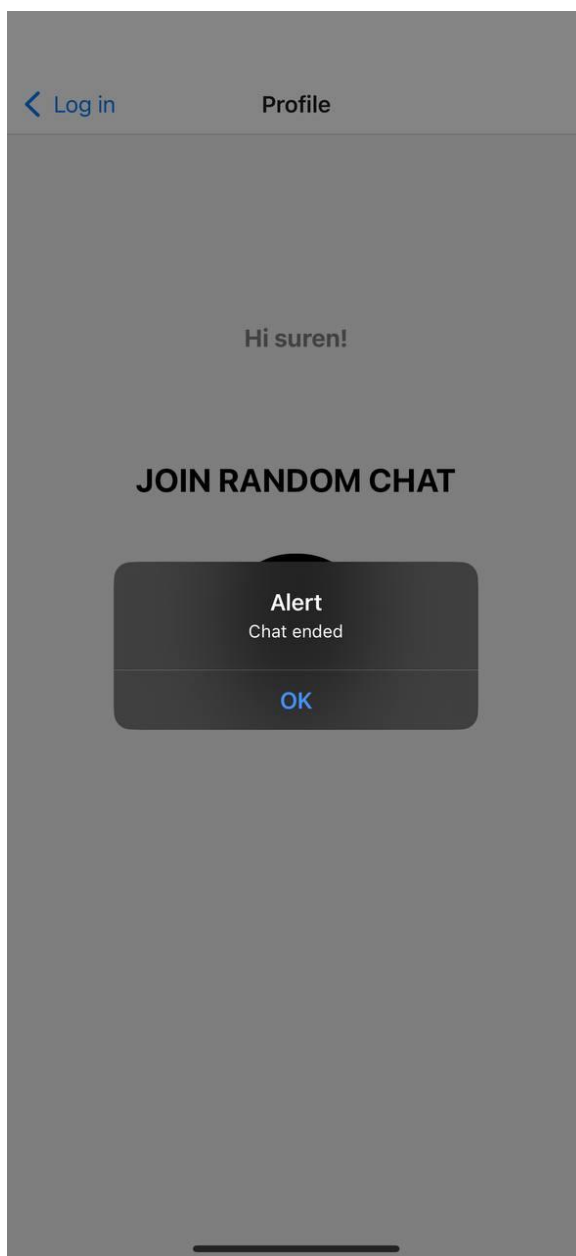


Рисунок 3.13 – Вихід з чату

3.3 Розробка серверної частини

Серверна частина – це компонент програмного забезпечення, який працює на сервері та відповідає за обробку даних, виконання логіки додатка та взаємодію з базою даних. Вона приймає запити від клієнтської частини, обробляє їх і відправляє відповідь назад на клієнт.

Серверна частина програми прихована від користувача. Прикладом серверної частини програми може бути реєстрація користувача. Коли користувач реєструється, він не бачить внутрішньої частини програми, такої

як запит до сервера і додавання даних до бази даних. Після виконання внутрішніх операцій сервер відправляє відповідь клієнту.

Основні функції серверної частини включають:

- обробка запитів;
- бізнес-логіка;
- комунікація;
- безпека.

Для створення серверної частини програми був використаний протокол HTTP(Hypertext Transfer Protocol). HTTP — це набір правил, за якими комп'ютери обмінюються даними в інтернеті. Він може передавати різноманітні формати файлів, такі як відео, аудіо та текст [17].

Протокол HTTP складається з двох елементів: клієнта та сервера. Клієнт відправляє запити до сервера, а сервер обробляє їх та повертає відповіді.

Процедуру відправки HTTP-запиту від клієнта продемонстровано на рисунку 3.4.

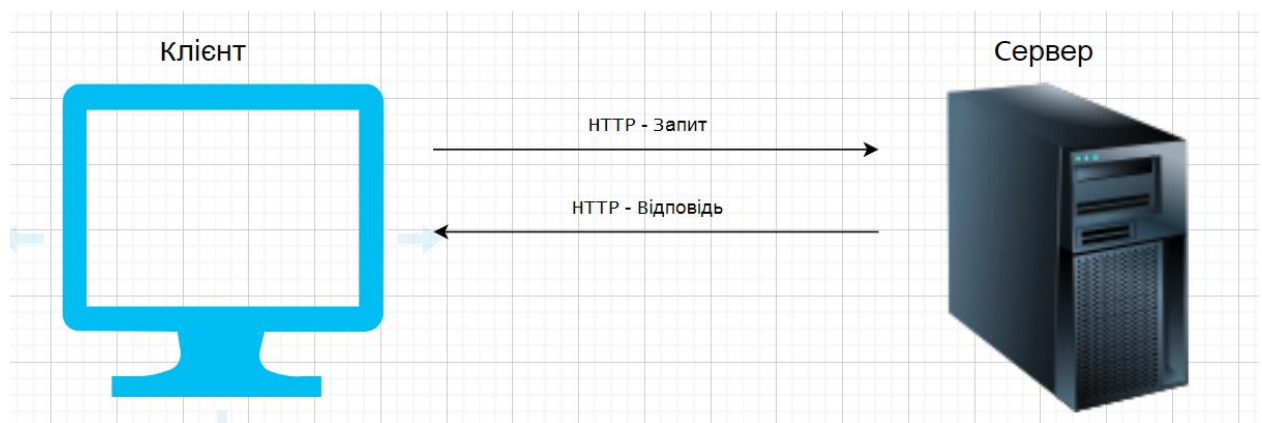


Рисунок 3.4 – Взаємодія клієнт та сервер

Клієнтом може бути будь-який пристрій, через який користувач запитує запит. Особливість клієнта – він завжди ініціює запит.

Сервер — це пристрій, який опрацьовує запити клієнта. Він може складатися як з одного комп'ютера, так і кластера.

Структура HTTP — повідомлення завжди однакова:

- метод HTTP – запиту – визначає набір методів запиту, що вказують на бажану дію, яка буде виконана для даного ресурсу;
- мета запиту визначається URL, який включає протокол, доменне ім'я та шлях до конкретного ресурсу на сервері;
- версія протоколу, що використовується (HTTP/1.1,HTTP/2), визначає структуру даних, які йдуть після стартового рядка.

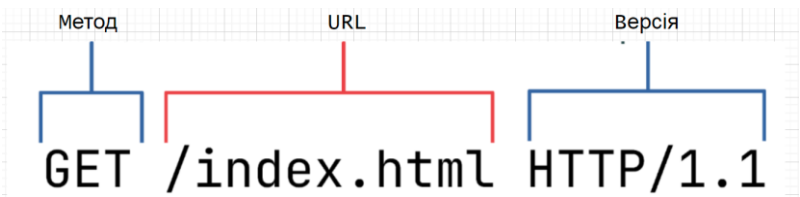


Рисунок 3.5 – Основні структурні елементи URL

Методи дозволяють визначити конкретну дію, яку клієнт запитує від сервера.

Таблиця 3.1 – Основні методи запитів

Метод	Опис
GET	Метод GET використовується для запиту даних із сервера без зміни його стану.
POST	Дозволяє надіслати дані на сервер.
HEAD	Запит з використанням цього методу зазвичай проводиться для того, щоб дізнатися розмір запитуваного ресурсу перед завантаженням.
PUT	Використовується для розміщення нових ресурсів на сервері.
DELETE	Видаляє наявні ресурси на сервері.
OPTIONS	Дозволяє запитати інформацію про

	сервер
PATCH	Дозволяє здійснити часткові зміни до вказаного ресурсу за заданим місцезнаходженням.

Отримання дозволу до ресурсу за протоколом HTTP здійснюється за допомогою вказівника URL.

Основні компоненти URL включають:

- протокол;
- доменне ім'я;
- порт;
- шлях;
- параметри запиту.

HTTP заголовки повідомляють серверу, як взаємодіяти та обробляти дані.

Таблиця 3.2 – Заголовки запитів

Заголовок	Опис
Host	Використовується для того, з якого конкретно хосту запитується ресурс.
Accept	Дозволяє вказати, який тип медіафайлів приймає клієнт
Authorization	Використовується як ще один метод ідентифікації клієнта на сервері.

Завершальна частина HTTP-запиту - це його тіло. Це частина HTTP запиту, яка містить дані, що надсилаються на сервер у вигляді тексту чи іншого формату.

HTTP-відповідь є повідомленням, яке сервер відправляє клієнту у відповідь на запит, отриманий від нього.

Структурі HTTP-відповіді: стартовий рядок, заголовки та тіло.

Рядок статусу(рисунок 3.6) в HTTP-відповіді є текстовим описом, який сервер надсилає клієнту разом із кодом стану, що вказує на результат обробки запиту.

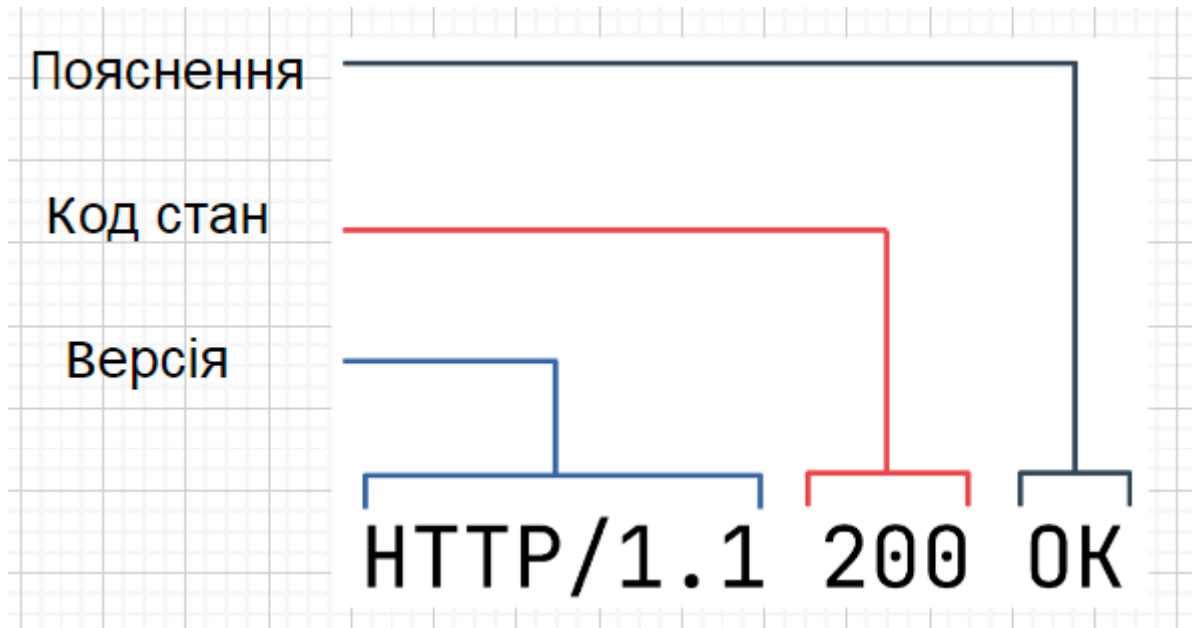


Рисунок 3.6 – Рядок стану відповіді.

Переваги HTTP:

- простота – за допомогою протоколу HTTP можна розробляти клієнтські програми;
- детальна документація – HTTP – найпопулярніший протокол в інтернеті. Таким чином HTTP докладно описаний різними мовами, і у документації є відповіді більшості питань;
- поширеність – HTTP вважається основним та універсальним, на ньому працюють практично всі сайти у світі;
- широкий спектр можливостей – у протоколі можна використовувати будь-які заголовки. Таким чином, функціонал протоколу збільшується за необхідності.

Недоліки HTTP:

- нема навігації – цей протокол немає гарної навігації серед ресурсів сервера;

- незахищеність – базовий HTTP не має шифрування;

- проблеми з розподіленими запитами;

Протокол HTTP був використаний при авторизації та реєстрації клієнта.

Всього було створено три маршрути з протоколом HTTP.

Для отримання ресурсів використовуйте метод `fetch()`, який є доступним як у глобальному контексті `Window`, так і в контекстах `Worker`. Він надає можливість отримати ресурси практично у будь-якому необхідному контексті.

Таблиця 3.3 – HTTP маршрути

Route	Method	Description
/sign-in	POST	Log in to profile
/sign-up	POST	Create a new profile
/find-random-chat	POST	Find a new random chat

При реєстрації користувача програма відправляє запит на сервер. Якщо такого користувача немає у базі даних, його заносять в базу і переадресовують на головну сторінку (рисунок 3.7). Якщо користувач вже зареєстрований на платформі, надсилається повідомлення.

При авторизації надсилається запит до бази даних. Якщо користувач існує, відбувається перехід на головний сайт. Якщо ж користувача немає, він може зареєструватися (рисунок 3.8).

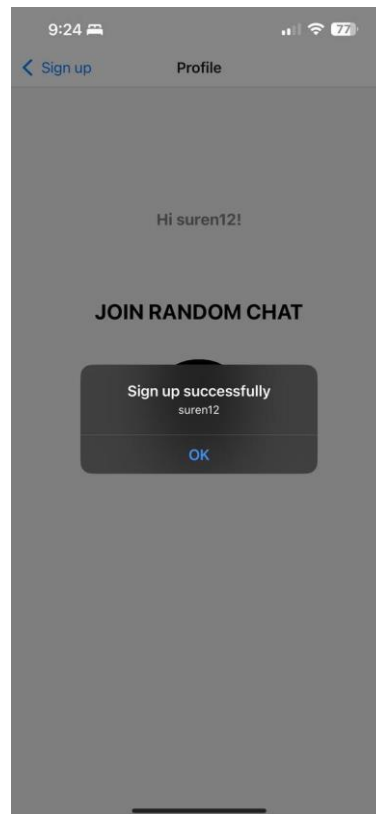


Рисунок 3.7 –Реєстрація пройшла успішна.

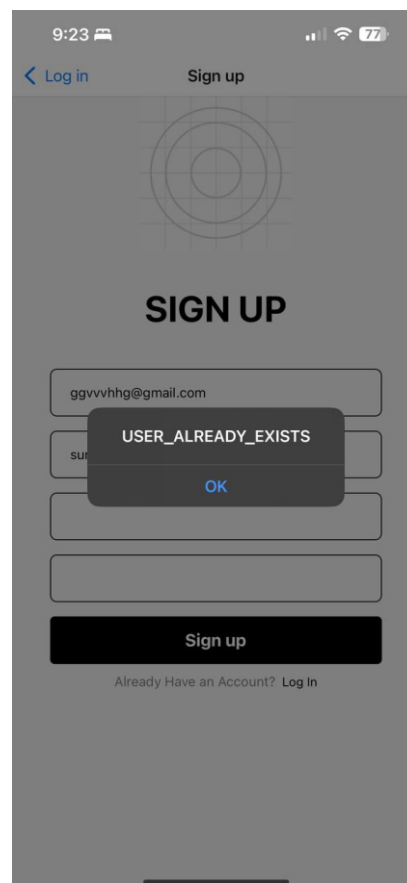
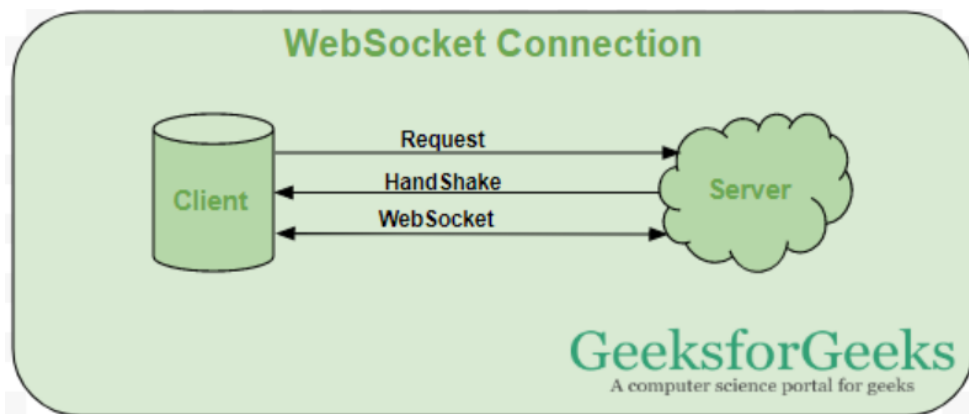


Рисунок 3.7 – Користувач існує у базі даних

Для реалізації двонаправленого зв'язку між клієнтським додатком і сервером використано WebSocket, який підтримує з'єднання для більш динамічного обміну даними в режимі реального часу. А також забезпечує кращу швидкість порівняно з HTTP запитам.

Приклад організації спілкування між клієнтом та сервером наведено на рисунку 3.8.



Рисунку 3.8 – Обмін повідомлень між клієнтом та сервером у WebSockets протоколі [19]

Переваги WebSocket:

- WebSocket – підтримує двонаправлений зв'язок між клієнтським додатком та сервером;
- WebSocket забезпечує низьку затримку при передачі даних, що працюють у режимі реального часу, таких як чати;
- WebSocket використовує менше трафіку в порівнянні з HTTP, оскільки не вимагає заголовків HTTP для кожного повідомлення;
- більшістю сучасних веб-браузерів та серверних платформ підтримують WebSocket.

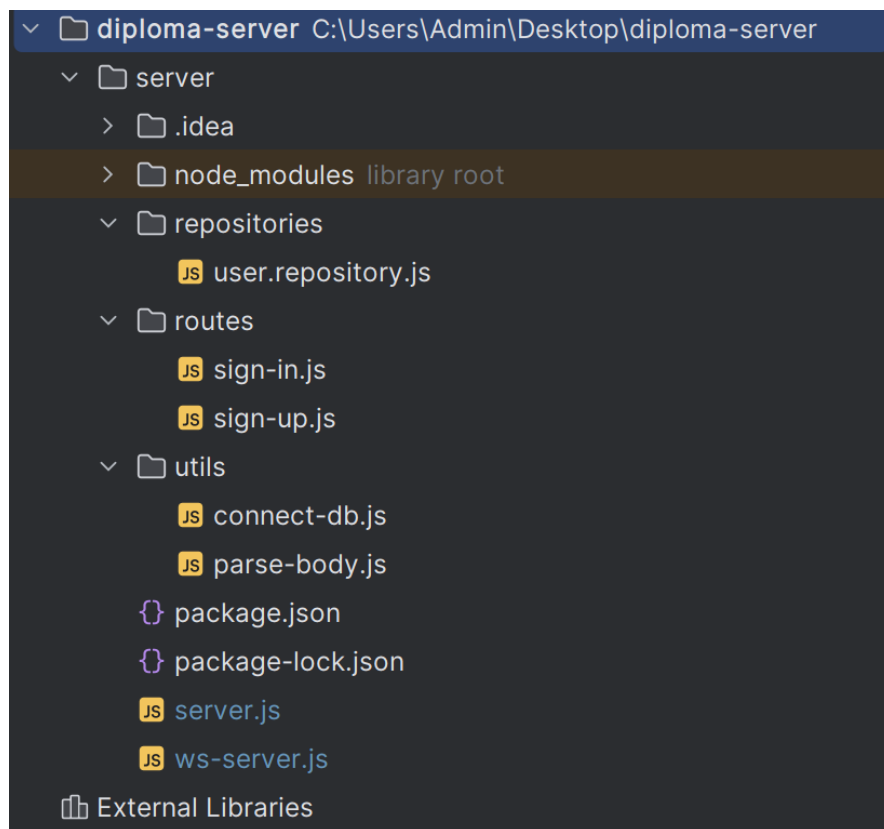
Недоліки WebSocket:

- не всі старі браузери підтримують WebSocket;
- складність реалізації ніж http-запитів;

– деякі проксі-сервери та брандмауери можуть блокувати або обмежувати його використання.

3.4 Розробка модуля підтримки зв'язку між співрозмовниками для анонімного чату

Для початкового встановлення з'єднання між клієнтом та сервером використовується протокол WebSocket. Структура додатку відображена на рисунку 3.9.



Рисунку 3.9 – Структура додатку серверної частини

У серверній частині використовуються дві технології. Перша технологія — це протокол HTTP, який дозволяє авторизуватися та реєструватися користувачеві завдяки надсиланню запитів на сервер. Друга технологія — WebSocket, яка дозволяє двосторонній зв'язок.

ВИСНОВОК

Під час виконання завдання було проведено огляд предметної області та сформульовано ключові функціональні та нефункціональні вимоги до розроблюваного продукту.

Для розробки додатку буде використані різні технології для розробки додатку, підключення до бази даних, використання фреймворку React Native, побудови графічного інтерфейсу а також проаналізовані різні API для збільшення можливості додатку.

Створений продукт буде корисний тим хто хоче поспілкуватися, але не розкривати свою конфіденційну інформацію.

ПЕРЕЛІК ДЖЕРЕЛ

1. Анонімний чат [Електронний ресурс]. – Режим доступу:
<https://lfyou.com.ua/about>
2. Yik Yak [Електронний ресурс]. – Режим доступу:
<https://www.webwise.ie/parents/explainer-what-is-yik-yak/>
3. Yik Yak [Електронний ресурс]. – Режим доступу:
<https://www.theverge.com/2017/4/28/15480052/yik-yak-shut-down-anonymous-messaging-app-square>
4. Omegle [Електронний ресурс]. – Режим доступу:
<https://uk.wikipedia.org/wiki/Omegle>
5. Omegle location locator [Електронний ресурс]. – Режим доступу:
<https://chromewebstore.google.com/detail/omegle-location-locator/bmgoghbncjflpjajddphcaghgknmjopo?pli=1>
6. What is Ask.fm? [Електронний ресурс]. – Режим доступу:
<https://webwise.ie/parents/explained-what-is-ask-fm-2/>
7. Unofficial Ask.fm App Now Available In Windows Phone Store [Електронний ресурс]. – Режим доступу: <https://mspoweruser.com/unofficial-ask-fm-app-now-available-in-windows-phone-store/>
8. Whisper [Електронний ресурс]. – Режим доступу:
https://play.google.com/store/apps/details?id=sh.whisper&hl=en_US
9. Chatous [Електронний ресурс]. – Режим доступу:
<https://talkwithstranger.com/chatsites/chatous>
10. Chatous [Електронний ресурс]. – Режим доступу:
https://play.google.com/store/apps/details?id=com.chatous.chatous&hl=en_US
11. Sarahah [Електронний ресурс]. – Режим доступу:
<https://en.wikipedia.org/wiki/Sarahah>
12. What is Sarahah and why was it removed from Google Play? [Електронний ресурс]. – Режим доступу:

<https://www.androidauthority.com/what-is-sarahah-790691/>

13. Kik — Messaging & Chat App [Электронный ресурс]. – Режим доступа: https://play.google.com/Kik%20%E2%80%94%20Messaging%20&%20Chat%20App/apps/details?id=kik.android&hl=en_US

14. Slack (software) [Электронный ресурс]. – Режим доступа: [https://en.wikipedia.org/wiki/Slack_\(software\)](https://en.wikipedia.org/wiki/Slack_(software))

15. Slack [Электронный ресурс]. – Режим доступа: https://play.google.com/store/apps/details?id=com.Slack&hl=en_US

16. chatgig [Электронный ресурс]. – Режим доступа: <https://www.chatgig.com/>

17. HTTP [Электронный ресурс] – Режим доступа: <https://skillbox.ru/media/code/chto-takoe-http-i-zachem-on-nuzhen/>

18. Metro Bundler : Under the hood [Электронный ресурс] – Режим доступа: <https://levelup.gitconnected.com/metro-bundler-under-the-hood-ffa254b5ca9c>

19. What is web socket and how it is different from the HTTP [Электронный ресурс] – Режим доступа: <https://geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/>

Додаток А

Зауваження нормоконтролера

Таблиця А.1 – Зауваження нормоконтролера

[illegible]

Додаток Б

Фрагменти лістинги програми

В.1 Реєстрація користувача (серверна частина)

```
export default {  
  path: "/sign-up",  
  method: "POST",  
  handler: (req, res, repositories) => {  
    const parsedBody = JSON.parse(req.body);  
    const { login, email, password } = parsedBody;  
  
    if (!login || !email || !password) {  
      res.statusCode = 400;  
      return { error: "INVALID_CREDENTIALS" };  
    }  
  
    const user = repositories.userRepository.getByLoginOrEmail(login, email);  
    if (user) {  
      res.statusCode = 400;  
      return { error: "USER_ALREADY_EXISTS" }  
    }  
  
    const createdUser = repositories.userRepository.insert({ login, email,  
password });  
    res.statusCode = 200;  
    return { id: createdUser.id, login: createdUser.login };  
  }  
}
```

В.2 Авторизація користувача (серверна частина)

```
export default {
  path: "/sign-in",
  method: "POST",
  handler: (req, res, repositories) => {
    const parsedBody = JSON.parse(req.body);
    const { login, password } = parsedBody;
    const user = repositories.userRepository.getByLoginAndPassword(login,
password);
    if (!user) {
      res.statusCode = 400;
      return { error: "USER_IS_NOT_FOUND" };
    }

    res.statusCode = 200;
    return { id: user.id, login: user.login };
  }
}
```

В 3 Звернення до даних клієнта (серверна частина)

```
export const users = [
  { id: 1, login: "suren", email: "dendy@dsad.com", password:
"qwerty123456789" },
  { id: 2, login: "ararat", email: "ararat@gmail.com", password:
"qwerty123456789" }
];

export const userRepository = () => ({
  getAll: () => {
    return users;
  }
})
```

```
},
```

```
getById: (id) => {
  return users.find((user) => user.id === id);
},
```

```
getByLogin: (login) => {
  return users.find((user) => user.login === login);
},
```

```
getByLoginAndPassword: (login, password) => {
  return users.find((user) => user.login === login && user.password ===
password);
},
```

```
getByLoginOrEmail: (login, email) => {
  return users.find((user) => user.login === login || user.email === email);
},
```

```
insert: (user) => {
  const newUser = { ...user, id: new Date() }
  users.push(newUser);
  return newUser;
}
})
```

В 4 сервере

```
import { createServer } from "node:http";
import signInRoute from "../routes/sign-in.js";
import signUpRoute from "../routes/sign-up.js";
import { parseBody } from "../utils/parse-body.js";
```

```

import {connectDb} from "./utils/connect-db.js";
import {userRepository} from "./repositories/user.repository.js";
import {startWebSocketServer} from "./ws-server.js";

// const hostname = "192.168.1.202"; Suren
const hostname = "192.168.1.202";
const port = 3000;
const wsPort = 8080;

const routes = [
  signInRoute,
  signUpRoute,
]

const db = await connectDb();

const repositories = {
  userRepository: userRepository(db),
}

const server = createServer(async (req, res) => {
  const path = req.url;
  const method = req.method;

  const route = routes.find((route) => route.path === path && route.method ===
method);
  if (!route) {
    console.log("Received unknown request");
    res.statusCode = 404;
    res.end();
  }
});

```

```

    return;
  }
  req.body = await parseBody(req);

  res.setHeader('Content-Type', 'application/json');
  const jsonResponse = route.handler(req, res, repositories);
  if (jsonResponse) {
    try {
      const response = JSON.stringify(jsonResponse);
      res.write(response);
    } catch (error) {
      console.error("Failed to transform response", { error });
    }
  }
  res.end();
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}`);
});

startWebSocketServer(wsPort, hostname);

```

B 5 WebSocket

```
import { WebSocketServer } from "ws";
import { userRepository } from "../repositories/user.repository.js";

/**
 * List of all active connected users
 * {{
 *   [userId]: connection
 * }}
 */
const connectedUsers = {};

/**
 * List of users who are waiting for random chat
 */
const usersSearchingChat = [];

/**
 * Active chats
 */
const activeChats = [];

/**
 *
 * @param message the message received as string in JSON format
 * @return {{
 *   event: string,
 *   payload: object,
 * }}
 */
```

```

*/
const parseEventMessage = (message) => {
  try {
    const parsedMessage = JSON.parse(message);
    if (!parsedMessage.event || typeof parsedMessage.event !== 'string') {
      console.error("Message does not contain event name", parsedMessage);
      throw new Error("Message does not contain event name");
    }
    parsedMessage.payload = parsedMessage.payload ? parsedMessage.payload :
  };
  return parsedMessage;
} catch (error) {
  console.error("Failed to parse event message", error);
  throw error;
}
}

const events = {
  /**
   * New user trying to establish connection with server
   * payload: { userId: number }
   */
  'establishConnection': ({ payload, connection }) => {
    const { userId } = payload;
    console.log(`User ${userId} is trying to establish connection with server`);
    if (!userId) {
      console.error("Error: userId is missing in the event establishConnection");
      return;
    }
    if (connectedUsers[userId]) {

```

```

        console.error("Error: User already has connection with server");
        return;
    }
    connectedUsers[userId] = connection;
    const user = userRepository().getById(userId);
    connection._customUserData = { userId, login: user.login };
    console.log(`User ${userId} established connection with server`);
},

/**
 * User sent a request to find a new random chat
 */
'findRandomChat': ({ senderId }) => {
    console.log(`User ${senderId} sent a request to find a new random chat`);
    if (!usersSearchingChat.length || !usersSearchingChat.includes(senderId)) {
        console.log(`User ${senderId} added to waiting list for random chat`);
        usersSearchingChat.push(senderId);
    }

    const randomUserIdIndex = usersSearchingChat.findIndex((userId) => userId
    !== senderId);
    const randomUserId = usersSearchingChat[randomUserIdIndex];
    if (!randomUserId) {
        console.log(`No users found for random chat for user ${senderId}`);
        return;
    }

    // Creating new chat for users
    activeChats.push({ users: [senderId, randomUserId] });
    const senderConnection = connectedUsers[senderId];

```

```

    const foundRandomUserConnection = connectedUsers[randomUserId];
    senderConnection.send(JSON.stringify({ event: 'randomChatFound' }));
    foundRandomUserConnection.send(JSON.stringify({ event:
'randomChatFound' }));

    senderConnection.send(JSON.stringify({
        event: 'newChatMessage',
        payload: { content: `Chat with
${foundRandomUserConnection._customUserData?.login} started`, type:
"system" }
    }));

    foundRandomUserConnection.send(JSON.stringify({
        event: 'newChatMessage', payload: { content: `Chat with
${senderConnection._customUserData?.login} started`, type: "system" }
    }));

    usersSearchingChat.splice(randomUserIdIndex, 1);

    const senderUserIdIndex = usersSearchingChat.findIndex((userId) => userId
=== senderId);
    usersSearchingChat.splice(senderUserIdIndex, 1);
    console.log(`Chat started between users ${senderId} and ${randomUserId}`);
},

/**
 * Received new chat message from user
 * payload: { }
 */
'newChatMessage': ({ payload, senderId }) => {
    const activeChat = activeChats.find((chat) => chat.users.includes(senderId));
    if (!activeChat) {
        console.error("Chat not found to send a new message");
    }
}

```

```

        return;
    }

    const receiverUserId = activeChat.users.find((userId) => userId !==
senderId);
    if (!receiverUserId) {
        console.error("Receiver not found for a new message");
        return;
    }

    const receiverUserConnection = connectedUsers[receiverUserId];
    if (!receiverUserConnection) {
        console.error("Receiver connection not found for a new message");
        return;
    }

    const senderUserConnection = connectedUsers[senderId];

    receiverUserConnection.send(JSON.stringify({ event: 'newChatMessage',
payload: { content: payload.content,
background:""#282830",user:senderUserConnection._customUserData?.login} }));
    senderUserConnection.send(JSON.stringify({ event: 'newChatMessage',
payload: { content: payload.content,
background:""#5258a3",user:senderUserConnection._customUserData?.login} }));
    console.log(`New chat message was sent: ${payload.content}`);
},

'leaveChat': ({ senderId }) => {
    const activeChatIndex = activeChats.findIndex((chat) =>
chat.users.includes(senderId));

```

```

const activeChat = activeChats[activeChatIndex];
if (!activeChat) {
  console.error("Chat not found to leave the chat");
  return;
}

const receiverUserId = activeChat.users.find((userId) => userId !==
senderId);
if (receiverUserId) {
  const receiverUserConnection = connectedUsers[receiverUserId];
  if (receiverUserConnection) {
    receiverUserConnection.send(JSON.stringify({ event: 'chatEnded' }));
  } else {
    console.error("Receiver connection not found when the chat was
ended");
  }
} else {
  console.error("Receiver not found when the chat was ended");
}

const senderUserConnection = connectedUsers[senderId];
if (senderUserConnection) {
  senderUserConnection.send(JSON.stringify({ event: 'chatEnded' }));
} else {
  console.error("Sender connection not found when the chat was ended");
}

activeChats.splice(activeChatIndex, 1);

const senderUserIdIndex = usersSearchingChat.findIndex((userId) => userId

```

```

=== senderId);
    if (senderUserIdIndex !== -1) {
        usersSearchingChat.splice(senderUserIdIndex, 1);
    }

    console.log(`${senderId} ended chat`);
}
};

export const startWebSocketServer = (port, host) => {
    const wss = new WebSocketServer({ host: host, port: port });

    wss.on("connection", (connection) => {
        connection.on('error', (error) => console.log("Failed to establish WS
connection", { error }));

        /**
         * Received new message from client, the message received as string in JSON
format
         *
         * message: {
         *   event: name of the event
         *   payload: payload of the event
         * }
         */
        connection.on('message', (rawMessage) => {
            const { event, payload } = parseEventMessage(rawMessage);
            const handler = events[event];
            if (!handler) {
                console.error(`Handler not found for event: ${event}`);
            }
        });
    });
};

```

```

        return;
    }
    const senderId = connection?._customUserData?.userId;
    const senderLogin = connection?._customUserData?.login;
    console.log(`Received event "${event}" from user ${senderId ||
'unknown'}`);

    handler({ connection, payload, senderId, senderLogin });

    console.log(`Processed event "${event}" from user "${senderId ||
'unknown'}" was successfully processed`);
    });

    /**
     * User connection was closed
     */
    connection.on('close', () => {
        const userId = connection?._customUserData?.userId;
        console.log(`Connection for user ${userId} closed`);

        events.leaveChat({ senderId: userId });
        delete connectedUsers[userId];
    });
});

console.log(`WebSocket server running at port ws://${host}:${port}`);
}

```

В 6 User Interface (Фронтальная часть)

```
import {StyleSheet, View, Pressable, Text} from "react-native";

export default function AppButton (props) {

  return (
    <View>
      <Pressable
        style={styles.button}
      >
        <Text
          style={styles.text}
        >
          {props.label}
        </Text>
      </Pressable>
    </View>
  )
}

const styles = StyleSheet.create({
  button: {
    padding:20,
    backgroundColor: "#1a34a9",
    borderRadius:8,
    marginLeft:"auto",
    marginRight:"auto",
```

```

    },
    find_chat:{
        backgroundColor: "#1a34a9",
    },
    button_exit:{
        width:"100%",
    },
    button_settings:{
        width:"100%",
    },

```

```

    text: {
        textAlign:"center",
        color: "white",
        fontSize:16,
        fontWeight:"bold",
    },
}))

```

```
/**
```

components

- ui - list of all User Interface components
- icons - list of all icons which is used in the app
- layouts - list of components which should wrap other component and set up the layout for them

```
*/
```

```

import {StyleSheet, TextInput, View} from "react-native";
export default function AppInput (props) {
  return (
    <TextInput
      value = {props.value}
      keyboardType = { props.typeField }
      secureTextEntry={ props.isSecure }
      style = { [styles.input, styles[props.styleInput] ? styles[props.styleInput]:
null ]}

      onChangeText = { props.eventHandler }
      defaultValue = { props.defaultValue }
      placeholder = { props.placeholder }
      placeholderTextColor={"grey"}
    />
  )
}

const styles = StyleSheet.create( {
  input: {
    backgroundColor:"#edf1fe",
    padding: 20,
    borderRadius: 8,
    width: "100%",
    borderWidth:.2,
  },
  errorField: {
    borderColor:"#dc2626",
    backgroundColor:"#fef2f2",
    borderWidth:1,
  },

```

```

    inputForText: {
      width:"80%"
    }
  })
import {StyleSheet} from "react-native";
import {Text} from "react-native";
export default function AppText(props) {
  return (
    <Text style={styles[props.style]}>
      { props.label }
    </Text>
  )
}

```

```

const styles = StyleSheet.create( {
  titleForm: {
    fontSize:32,
    fontWeight:"bold",
    color:"blue",
    textAlign:"center",
  },
  descriptionForm: {
    fontSize:16,
    textAlign:"center",
    padding:30,
  },
  isUserHaveAccount: {
    marginTop:26,
    marginBottom:26,
    textAlign:"center",

```

```

        fontWeight:"bold",
    },
    userContinueWithSocial: {
        color:"blue",
        textAlign:"center",
        fontWeight:"600",
        marginBottom:10,
    },
    errorMessage: {
        color:"red",
        fontSize:14,
        marginTop:6,
    }
})

```

В 7 Повідомлення на помилки

```

export const ErrorMessages = {
    ERROR_REQUIRED: "Required field.",
    ERROR_LENGTH_USER_NAME: "Minimum user name length is 5",
    ERROR_LENGTH: "Minimum password length is 11",
    ERROR_PASSWORD:"Incorrectly format Password",
    ERROR_FORMAT_USER_NAME: "Incorrectly entered user name.",
    ERROR_FORMAT_EMAIL: "Incorrectly entered Email.",
    ERROR_NOT_SAME_PASSWORD: "Please make sure passwords match.",
};

```

В 8 Заголовки полів

```

export const formLabels = {
    TITLE_CREATE_ACCOUNT: "Create account",
    TITLE_AUTHORIZATION: "Login here",
    LABEL_DESCRIPTION_REGISTRATION: "Create an account so you can
explore all the existing jobs",

```

```

    LABEL_DESCRIPTION_AUTHORIZATION: "Welcome back you've been",
    LABEL_CREATE_NEW_ACCOUNT: "Create new account",
    LABEL_HAS_ACCOUNT: "Already have an account",
  }

```

В 9 Валідація форми

```

import {ErrorMessage} from "../error-messages";

const EMAIL_REGEX_RULE = /^w*@[a-z,A-Z]*\.[a-z,A-Z]{2,}/gm;
const USER_NAME_RULE = /^[a-zA-Z1-9]+$/gm;
const PASSWORD_NAME_RULE = /^[a-zA-Z1-9]+$/gm;
const PASSWORD_MIN_LENGTH = 11;

export const validateUserEmail = (value) => {
  if (!value) {
    return ErrorMessage.ERROR_REQUIRED;
  } else if (!value.match(EMAIL_REGEX_RULE)){
    return ErrorMessage.ERROR_FORMAT_EMAIL;
  }
}

export const validateUserName = (value) => {
  if (!value) {
    return ErrorMessage.ERROR_REQUIRED;
  } else if(value.length > 25 || value.length < 4){
    return ErrorMessage.ERROR_LENGTH_USER_NAME;
  } else if(!value.match(USER_NAME_RULE)){
    return ErrorMessage.ERROR_FORMAT_USER_NAME;
  }
}

```

```

export const validateUserPassword = (value) => {
  if (!value) {
    return ErrorMessages.ERROR_REQUIRED;
  } else if (!value.match(PASSWORD_NAME_RULE)){
    return ErrorMessages.ERROR_PASSWORD
  } else if (value.length < PASSWORD_MIN_LENGTH){
    return ErrorMessages.ERROR_LENGTH;
  }
}

```

```

export const validateUserConfirmPassword = (password,confirmPassword) => {
  if (!confirmPassword) {
    return ErrorMessages.ERROR_REQUIRED;
  } else if (password !== confirmPassword ){
    return ErrorMessages.ERROR_NOT_SAME_PASSWORD;
  }
}

```

```

export const formIsValid = (validationResult) => {
  return Object.values(validationResult).filter((value) => value !==
undefined).length === 0;
}

```

B 9 WebSocketProvider

```

import {Alert} from "react-native";
import {createContext, useContext, useEffect, useRef, useState} from "react";

const WebSocketContext = createContext(null);
export const WebSocketProvider = ({ children }) => {
  const socket = useRef(null);
  const [connected, setConnected] = useState(false);
  useEffect(() => {
    socket.current = new WebSocket('ws://192.168.1.202:8080');

    socket.current.onopen = () => {
      console.log('WebSocket connection opened');
      setConnected(true);
    }

    socket.current.onclose = () => {
      console.log('Disconnected from WebSocket');
    };

    socket.current.onerror = (error) => {
      console.error('WebSocket error:', error);
    };

    return () => {
      socket.current.close();
    };
  }, []);

```

```

    return (
      <WebSocketContext.Provider value={socket}>
        {children}
      </WebSocketContext.Provider>
    );
  };
};

```

```

export const useWebSocket = () => useContext(WebSocketContext);

```

B 11 КОМПОНЕНТ UserHomeView

```

import React, {useEffect, useRef, useState} from 'react'
import {Alert, Pressable, StyleSheet, Text, TouchableOpacity, View} from 'react-native'
import {useWebSocket} from "../utils/WebSocketProvider";

```

```

export default function UserHomeView({navigation, route}){
  const socket = useWebSocket();

  useEffect(() => {
    if (socket.current) {
      const message = {
        event: 'establishConnection',
        payload: { userId: route.params.userId },
      };
      socket.current.send(JSON.stringify(message));
    }
  }, []);

  return(

```

```

<View style={styles.container}>
  <Text style={styles.greeting}>Hi {route.params.username}!</Text>
  <Text style={styles.title}>Join random chat</Text>
  <TouchableOpacity style={styles.buttonView} onPress={() => {
    navigation.navigate('SearchingChat', {username:
route.params.username});
  }}>
    <Text style={styles.buttonText}>Go</Text>
  </TouchableOpacity>
</View>
)
}

```

```

const styles = StyleSheet.create({
  container : {
    alignItems : "center",
    paddingTop: 100,
  },
  greeting : {
    fontSize : 18,
    fontWeight : "bold",
    textAlign: "center",
    paddingVertical : 40,
    color : "grey"
  },
  title : {
    fontSize : 24,
    fontWeight : "bold",
    textTransform : "uppercase",
    textAlign: "center",

```

```

paddingVertical : 40,
color : "black"
},
buttonView: {
  backgroundColor: "black",
  padding: 40,
  borderRadius: 100,
},
buttonText: {
  color: "white",
  fontSize: 24,
  fontWeight: "bold",
}
})

```

B 12 КОМПОНЕНТ SignUpView

```

import React, {useCallback, useState} from 'react'
import { Alert, Button, Image, Pressable, SafeAreaView, StyleSheet, Switch, Text,
TextInput, View } from 'react-native'
import {
  formIsValid,
  validateUserConfirmPassword,
  validateUserEmail,
  validateUserName,
  validateUserPassword
} from "../utils/validation-utils";
import AppText from "../components/ui/AppText/AppText";
const logo = require("../assets/icon.png")
const signUp = async (login,email,password) =>{
  const response = await fetch("http://192.168.1.202:3000/sign-up", {
    method: 'POST',

```

```

    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ login,email, password })
  });
const jsonResponse = await response.json();
return jsonResponse;
}

export default function SignUpView({navigation}){
  const [email,setEmail]= useState("ararat@gmail.com");
  const [username,setUsername]= useState("ararat");
  const [confirmPassword,setConfirmPassword]= useState("qwerty123456789");
  const [password,setPassword]= useState("qwerty123456789");
  const [validationResult, setValidationResult] = useState({});

  const onSubmit = useCallback( async () => {

    const response = await signUp(username,email,password);
    if(response.error){
      Alert.alert(response.error)
      return;
    }
    Alert.alert("Sign up successfully",`${username}`)
    navigation.navigate("UserHome",{username})
  }, [email,username, password]);

  const submitForm = () => {
    const validationResult = {
      login: validateUserName(username),
      password: validateUserPassword(password),

```

```

        confirmPassword:
validateUserConfirmPassword(password,confirmPassword),
        email:validateUserEmail(email),
    };
    const validForm = formIsValid(validationResult)
    if (validForm) {
        onSubmit();
        return
    }
    setValidationResult(validationResult);
}
return(
    <SafeAreaView style={styles.container}>
        <Image source={logo} style={styles.image} resizeMode='contain' />
        <Text style={styles.title}>Sign up</Text>
        <View style={styles.inputView}>
            <View>
                <TextInput
                    style={styles.input}
                    placeholder='Email'
                    value={email}
                    onChangeText={setEmail}
                    autoCorrect={false}
                    autoCapitalize='none'
                />
                {validationResult.email
                    ? <AppText label={validationResult.email} style="errorMessage"/>
                    : null
                }
            </View>
        </View>
    )

```

```

<View>
  <TextInput
    style={styles.input}
    placeholder='Username'
    value={username}
    onChangeText={setUsername}
    autoCorrect={false}
    autoCapitalize='none'
  />
  {validationResult.login
    ? <AppText label={validationResult.login} style="errorMessage"/>
    : null
  }
</View>
<View>
  <TextInput
    style={styles.input}
    placeholder='Password'
    secureTextEntry
    value={password}
    onChangeText={setPassword}
    autoCorrect={false}
    autoCapitalize='none'
  />
  {validationResult.password
    ? <AppText label={validationResult.password}
style="errorMessage"/>
    : null
  }
</View>

```

```

<View>
  <TextInput
    style={styles.input}
    placeholder='Confirm Password'
    secureTextEntry
    value={confirmPassword}
    onChangeText={setConfirmPassword}
    autoCorrect={false}
    autoCapitalize='none'
  />
  {validationResult.confirmPassword
    ? <AppText label={validationResult.confirmPassword}
      style="errorMessage"/>
    : null
  }
</View>
</View>

<View style={styles.buttonView}>
  <Pressable style={styles.button} onPress={() => submitForm()}>
    <Text style={styles.buttonText}>Sign up</Text>
  </Pressable>
</View>

  <Text style={styles.footerText}>Already Have an Account?<Text
style={styles.signup} onPress={() => {
  navigation.navigate('LogIn', {name: 'Jane'})
}}> Log In</Text></Text>
</SafeAreaView>
)

```

```
}
```

```
const styles = StyleSheet.create({  
  container : {  
    alignItems : "center",  
    paddingTop: 100,  
  },  
  image : {  
    height : 160,  
    width : 170  
  },  
  title : {  
    fontSize : 38,  
    fontWeight : "bold",  
    textTransform : "uppercase",  
    textAlign: "center",  
    paddingVertical : 40,  
    color : "black"  
  },  
  inputView : {  
    gap : 15,  
    width : "100%",  
    paddingHorizontal : 40,  
    marginBottom : 5  
  },  
  input : {  
    height : 50,  
    paddingHorizontal : 20,  
    borderColor : "black",  
    borderWidth : 1,  
  },  
})
```

```

        borderRadius: 7
    },
    button : {
        backgroundColor : "black",
        height : 50,
        borderColor : "black",
        borderWidth : 1,
        marginTop: 10,
        borderRadius : 5,
        alignItems : "center",
        justifyContent : "center"
    },
    buttonText : {
        color : "white" ,
        fontSize: 18,
        fontWeight : "bold"
    },
    buttonView :{
        width : "100%",
        paddingHorizontal : 40,
        marginBottom: 10
    },
    optionsText : {
        textAlign : "center",
        paddingVertical : 10,
        color : "gray",
        fontSize : 13,
        marginBottom : 6
    },
    icons : {

```

```

        width : 40,
        height: 40,
      },
      footerText : {
        textAlign: "center",
        color : "gray",
      },
      signup : {
        color : "black",
        fontSize : 13
      }
    })

```

В 13 Пошук чату

```

import React, {useEffect, useState} from 'react'
import {Pressable, StyleSheet, Text, TouchableOpacity, View} from 'react-native'
import {useWebSocket} from "../utils/WebSocketProvider";

```

```

export default function SearchingChatView({navigation, route}){
  const socket = useWebSocket();

  useEffect(() => {
    socket.current.send(JSON.stringify({ event:"findRandomChat"} ));

    socket.current.onmessage = (message) => {
      const { event } = JSON.parse(message.data);
      console.log("Received event", event);

      if (event === "randomChatFound") {

```

```

        navigation.navigate("Chat", {
            username: route.params.username,

        })
    }
};
}, []);

return(
    <View style={styles.container}>
        <Text style={styles.title}>Looking the best chat for you..</Text>
        <Text style={styles.emoji}>☐</Text>
    </View>
)
}

const styles = StyleSheet.create({
    container : {
        alignItems : "center",
        paddingTop: 150,
    },
    title : {
        fontSize : 22,
        textAlign: "center",
        paddingVertical : 40,
        color : "grey"
    },
    emoji: {
        fontSize : 78,
    },

```

```
})
```

B 14 КОМПОНЕНТ ЧАТУ

```
import React, {useEffect, useRef, useState} from 'react'
import { useWebSocket } from '../utils/WebSocketProvider';
import {
  Alert, Button,
  Dimensions, FlatList,
  KeyboardAvoidingView,
  Pressable,
  ScrollView,
  StyleSheet,
  Text,
  TextInput,
  TouchableOpacity,
  View
} from 'react-native'
import AppInput from "../components/ui/AppInput/AppInput";
import AppButton from "../components/ui/AppButton/AppButton";

export default function SearchingChatView({navigation, route}){
  const [listMessages, setNewMessagesInStorage] = useState([]);

  const [message, setMessage] = useState("");
  const scrollView = useRef(null);

  const socket = useWebSocket();
  function updateMessage (messageUser){
    setMessage(messageUser)
```

```

    }
    const scrollBottom = () => {
      if (scrollView.current) {
        scrollView.current.scrollToEnd({ animated: true });
      }
    };

    socket.current.onmessage = (message) => {
      const { event, payload } = JSON.parse(message.data);
      console.log("Received event", { event, payload });

      if (event === "newChatMessage") {
        setNewMessagesInStorage([...listMessages, { content: payload.content,
type:payload.type, background: payload.background,
user:payload.user,time:displayTime()}}])
        return;
      }

      if (event === "chatEnded") {
        navigation.navigate("UserHome", { ...route.params });
        alert("Chat ended");
      }
    };

    const sendMessage = async () => {
      socket.current.send(JSON.stringify({
        event: "newChatMessage",
        payload: { content: message }
      }));
      setMessage("")
    }
  }
}

```

```

}
function displayTime() {
  const now = new Date();
  let hours = now.getHours();
  let minutes = now.getMinutes();
  if (hours < 10){
    hours = '0' + hours;
  }
  if (minutes < 10){
    minutes = '0' + minutes;
  }
  return hours + ':' + minutes;
}

const renderListMessage = (message) =>{
  console.log(message)
  return(
    <View style={message.type === "system" ? [styles.connectedUser]:
[styles.message,{backgroundColor:message.background}]} >
      <Text style={{color:'white'}}>
        {message.type !== "system" ? message.user +': ' + message.content:
message.content}
      </Text>
      {
        message.type !== "system" ?
          <Text style={{color:"white"}}>
            {message.time}
          </Text>: ""
      }
    )
  }

```

```

</View>

)
}
return(
  <View style={styles.containerChat}>
    <ScrollView
      contentContainerStyle={styles.containerMessage}
      ref={scrollView}
      onContentSizeChange={scrollBottom}
    >
      {listMessages.map((item)=> renderListMessage(item))}
    </ScrollView >
    <KeyboardAvoidingView behavior={"padding"}
      keyboardVerticalOffset={100} style={styles.containerWriteMessage} >
      <AppInput
        placeholder="Write a message..."
        value={message}
        id="inputForWriteMessage"
        styleInput="inputForText"
        eventHandler={(messageUser) => {
          updateMessage(messageUser)
        }}
      />
      <AppButton
        label="Send"
        eventHandler={sendMessage}
      />
    </KeyboardAvoidingView>

```

```

    </View>
  )
}

```

```

const styles = StyleSheet.create({
  containerChat: {
    flex: 1,
    justifyContent: 'space-between',
  },
  containerText:{
    textAlign:"left",
  },
  containerWriteMessage: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    marginBottom:10,
    backgroundColor: '#f0f0f0',
  },
  containerMessage: {
    flexGrow: 1,
    justifyContent: 'flex-end',
  },
  connectedUser:{
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor:"#289DCCFF",
    color:"white",
  },

```

```

padding:12,
marginBottom:10,
},

message: {
padding:12,
marginBottom:10,
flexDirection: 'row',
alignItems: 'center',
justifyContent: 'space-between',
borderRadius:15,
color:"white",
},
})

```

В 15 Головний файл

```

import {NavigationContainer} from "@react-navigation/native";
import {createNativeStackNavigator} from '@react-navigation/native-stack';
import HomeView from "./views/HomeView";
import SignUpView from "./views/SignUpView";
import LoginView from "./views/LoginView";
import UserHomeView from "./views/UserHomeView";
import SearchingChatView from "./views/SearchingChatView";
import ChatView from "./views/ChatView";
import {useWebSocket, WebSocketProvider} from "./utils/WebSocketProvider";
import {Button, TouchableOpacity,Text} from "react-native";
import navigationContainer from "@react-
navigation/native/src/NavigationContainer";
import {useEffect, useRef} from "react";

```

```

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <WebSocketProvider>
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeView}
          options={{title: 'Let\'s chat'}}
        />
        <Stack.Screen name="LogIn" component={LoginView} options={{title:
'Log in'}}/>
        <Stack.Screen name="SignUp" component={SignUpView}
options={{title: 'Sign up'}}/>
        <Stack.Screen name="UserHome" component={UserHomeView}
options={{title: 'Profile'}}/>
        <Stack.Screen name="SearchingChat" component={SearchingChatView}
options={{title: 'Searching chat..'}}/>
        <Stack.Screen
          name="Chat"
          component={ChatView}
          options={{({ navigation,route }) => ({
            title: "Chat",
            headerTitleAlign:"center",
            headerBackButtonMenuEnabled: false,
            gesturesEnabled: false,

```

```

        headerLeft: () => (
            <ExitChat navigation={navigation} route={route} />
        ),
    )}
    />
</Stack.Navigator>
</NavigationContainer>
</WebSocketProvider>
);
}

const ExitChat = ({ navigation, route }) => {
    const socket = useWebSocket();
    return (
        <TouchableOpacity style={{ backgroundColor: "white" }} onPress={() => {
            navigation.goBack();
            socket.current.send(JSON.stringify({ event: "leaveChat" }));
        }}>
            <Text style={{ fontSize: 17, fontWeight: 500, color: '#289DCCFF' }}>End
Chat</Text>
        </TouchableOpacity>

    );
};

```

В 16 КОМПОНЕНТ HomeView

```

import React from 'react'
import { Pressable, StyleSheet, Text, TouchableOpacity, View } from 'react-native'

export default function HomeView({ navigation }) {

```

```

return(
  <View style={styles.container}>
    <Text style={styles.title}>Let's chat!</Text>
    <TouchableOpacity style={styles.buttonView} onPress={() => {
      navigation.navigate('LogIn', {name: 'Jane'});
    }}>
      <Pressable>
        <Text style={styles.buttonText}>Go</Text>
      </Pressable>
    </TouchableOpacity>
  </View>
)
}

```

```

const styles = StyleSheet.create({
  container : {
    alignItems : "center",
    paddingTop: 100,
  },
  title : {
    fontSize : 38,
    fontWeight : "bold",
    textTransform : "uppercase",
    textAlign: "center",
    paddingVertical : 40,
    color : "black"
  },
  buttonView: {
    backgroundColor: "black",
    padding: 40,

```

```
        borderRadius: 100,  
    },  
    buttonText: {  
        color: "white",  
        fontSize: 24,  
        fontWeight: "bold",  
    }  
})
```