Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Розробка клієнт-серверного веб-застосунку для спілкування між користувачами

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-6
В’ячеслав СОЧІВКІН
(власне ім’я, прізвище)

Спеціальність
123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Антон ГАВРАШЕНКО
(посада, власне ім’я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти Перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Сочівкіну В’ячеславу Михайловичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка клієнт-серверного веб-застосунку для спілкування між користувачами

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) архітектурна модель: клієнт-серверна з використанням SPA (Single-Page Application); 2) мережевий протокол: основний – WebSocket, допоміжний - HTTPS (REST API); 3) модель передачі даних: JSON (JavaScript Object Notation); 4) стек протоколів: TCP/IP з підтримкою SSL/TLS для захищеної передачі 5) механізм автентифікації: JWT (JSON Web Tokens)

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз проблеми та огляд існуючих рішень
2) вибір технології розробки та інструментальних засобів
3) розробка програмних модулів
4) відлагодження програмних модулів
5) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

Слайд-презентація – 10 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.25 – 31.05.25	
2	Вибір технології розробки та інструменталь-	01.05.25 – 03.06.25	
	них засобів		
3	Розробка та відлагодження програмного	04.06.25 – 09.06.25	
	забезпечення		
4	Оформлення матеріалів кваліфікаційної роботи	10.06.25 – 11.06.25	
5	Подання кваліфікаційної роботи керівнику та	12.06.25 – 13.06.25	
	її попередній захист		
6	Подання кваліфікаційної роботи на	14.06.25 – 15.06.25	
	рецензування		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Антон ГАВРАШЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 80 с., 33 рис., 2 табл., 2 дод., 30 джерел.

ПОВІДОМЛЕННЯ, ПРИСТРІЙ, ВЕБЗАСТОСУНОК, REACT, КЛІЄНТ, СЕРВЕР, КОРИСТУВАЧ, МЕСЕНДЖЕР, ПЕРЕДАЧА ДАНИХ, БРАУЗЕР.

Метою кваліфікаційної роботи є розробка клієнт-серверного вебзастосунку, який призначений для спілкування користувачів між собою за допомогою повідомлень та дзвінків.

У ході виконання кваліфікаційної роботи було проаналізовано основні виклики та проблеми розробки таких застосунків, те, як їх вирішують або знаходять для них компроміс в існуючих програмних рішеннях, поставлено конкретні задачі для розробки власного продукту, вибрано напрям і стек програмування, зокрема було обгрунтовано основні причини вибору технологій та можливі їх аналоги. Також було успішно реалізовано запланований функціонал, такий як повідомлення, дзвінки, взаємодія з ними, персоналізація, а також багато фундаментальних речей, які роблять функціонування застосунку стабільним. Було протестовано його, визначено системні вимоги та роботу застосунку в різних браузерах. Також для застосунку було складено інструкцію користувача, яка дозволить навіть необізнаним в технологіях людям успішно використовувати месенджер, не виходячи з зони комфорту.

ABSTRACT

Bachelor’s thesis: 80 pages, 33 figures, 2 tables, 2 appendices, 30 sources.

MESSAGE, DEVICE, WEB APPLICATION, REACT, CLIENT, SERVER, USER, MESSENGER, DATA TRANSFER, BROWSER.

The major goal of this thesis is the development of a client-server web application that is designed for users to communicate with each other using messages and calls.

During the thesis, the main challenges and problems of developing such applications were analyzed, how they are solved or a compromise is found for them in existing software solutions, specific tasks were set for developing our own product, the direction and programming stack were chosen, in particular, the main reasons for choosing technologies and their possible analogues were substantiated. The planned functionality was also successfully implemented, such as messages, calls, interaction with them, personalization, as well as many fundamental things that make the application stable. It was tested, the system requirements and the operation of the application in different browsers were determined. Also, a user manual was compiled for the application, which will allow even people who are not familiar with technology to successfully use the messenger without leaving their comfort zone.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

Месенджер – програмний застосунок, що використовується для обміну повідомленнями між користувачами (від англ. Messenger – зв’язківець)

UI – користувацький інтерфейс (від англ. User Interface)

UX – досвід користувача (від англ. User eXperience)

HTML – мова розмітки гіпертексту (від англ. HyperText Markup Language)

CSS – мова опису зовнішнього вигляду стилів (від англ. Cascading Style Sheets)

React – бібліотека з відкритим вихідним кодом для мов програмування JavaScript або TypeScript, що створена для розробки користувацьких інтерфейсів

Аутентифікація – процедура, що відповідає за перевірку на справжність користувача та/або введених ним даних

Мультимедіа – комбінування різних форм представлення інформації на одному носієві

NFT – невзаємнозаміняємий токен (від англ. Non-Fungible Token)

CDN – мережа з постачання вмісту (від англ. Content Delivery Network)

HTTPS – захищений протокол з передачі гіпертексту (від англ. HyperText Transfer Protocol Secure).

DOM – об’єктна модель документів. Є фундаментальним представленням HTML-коду (від англ. Document Object Model)

Markdown – мова розмітки, призначена для стилізації текстових файлів

Браузер – застосунок для перегляду інтернет-сторінок, а також файлів пристрою та їх каталогів (від англ. browser – переглядач)

Рушій – програмне або апаратне забезпечення для розробки застосунку

Тег – одна з множин типів даних HTML, що характеризується ключовими символами або словами (від англ. tag – бірка, ярлик)

Стиль – множина, якою класифікуються параметри, які можна змінювати в CSS

Система хмарного зберігання – мережа на основі віддаленого сервера, в якій можна зберігати файли користувачів замість фізичного простору на диску

NPM – вузловий менеджер пакетів (від англ. Node Package Manager)

CLI – інтерфейс командного рядка (від англ. Command Line Interface)

Персоналізація – процес налаштування користувачем системи під себе на основі наданих інструментів

MVP – мінімальний життєздатний продукт (від англ. Minimal Viable Product)

Ендпоінт – звернення до маршруту HTTP або HTTPS-методом (від англ. Endpoint – кінцева точка)

Плейсхолдер – дані, як текстові, так і мультимедіа, що призначені для заповнення наявного місця (від англ. Placeholder – тримач місця)

WebRTC – технологія для комунікації між користувачами в реальному часі (від англ. Web Real-Time Communications)

P2P – мережева архітектура, де учасники виклику напряму пов’язані між собою (від англ. Peer-to-Peer)

PiP – режим відображення, що створює міні-вікно поверх інших (від англ. Picture-in-Picture – картинка в картинці)

QoL – зміна чи нововведення, призначена для покращення користування застосунком (від англ. Quality of Life – якість життя)

Медіазапити – набір запитів у CSS, що дозволяє по-різному стилізувати дані в залежності від різних технічних параметрів пристрою користувача

ВСТУП

Люди – соціальні істоти, які прагнуть взаємодії з іншими. Цей процес триває протягом усієї історії людства. Ми намагалися передавати інформацію з початку існування, спочатку за допомогою жестів, потім усно і, нарешті, письмово. Цифрові засоби обміну повідомленнями, які зараз є частиною нашого повсякденного життя, ознаменували сучасний етап розвитку комунікації.

Одним із найважливіших аспектів нашого існування є передача інформації через повідомлення. Вони дозволяють нам вирішувати складні проблеми, будувати співпрацю та обмінюватися знаннями, а також дозволяють нам обмінюватися ідеями, емоціями та враженнями. Повідомлення, які можуть варіюватися від простих послідовностей символів до складних логічних конструкцій, є інструментом, що дозволяє нам взаємодіяти з реальністю та структурувати її.

У минулому передача інформації часто була повільною. Для цього використовувалися спеціально навчені люди, дим, сигнальні вогні, птахи, особливо голуби, і згодом поштові служби. Хоча ці методи були певною мірою успішними, вони не завжди були надійними, оскільки повідомлення могло бути прочитане неавторизованими особами, воно могло не дійти до одержувача або втратити свою актуальність.

З технологічним прогресом, інформація стала передаватись між розмовниками швидше і надійніше. У підсумку прогрес призвів до появи месенджерів – спеціалізованих програмних продуктів, що сприяють швидкому, легкому та доступному спілкуванню в цифровому середовищі.

Сучасні месенджери стали неймовірно важливим винаходом для людства, і майже кожен користувач знайде те, що буде робити в ньому: хтось буде спілкуватись зі своїми знайомими, а хтось – використовувати їх для своєї роботи або навчання.

Кількість функцій у сучасних месенджерах стабільно збільшується, а самі вони вже встигли зазнати еволюції і стали не просто інструментом для зв’язку, а фактично повноцінною соціальною мережею. Багато спільнот в месенджерах переросли у повноцінні тематичні форуми або новинні джерела. Під впливом таких нововведень у побутовій мові активно з’являються нові терміни, такі як «телеграм-канал». Це підтверджує той факт, що месенджери вже невід'ємно пов'язані з людством

Месенджери настільки чудово виконують свою роль, що вже давно стали важливим інструментом сучасного життя, який буде корисними і в приватному, і в професійному середовищі, і в освіті. Вони дозволяють користувачам миттєво, надійно і на будь-яку відстань надсилати та отримувати повідомлення, здійснювати дзвінки та переміщувати файли, зображення, відео та інші типи даних. Месенджери стали частиною цифрового життя завдяки своїй функціональності, гнучкості та простоті використання.

Створення власного месенджера – це складне та захопливе завдання, яке для досягнення бажаного результату вимагає досвіду в дизайні, програмуванні, алгоритмах та комунікаційних технологіях. Такий проєкт не лише дозволяє застосувати отримані знання на практиці, але й допомагає зрозуміти внутрішню роботу сучасних цифрових комунікаційних інструментів, які є вирішальними для кожного з нас у повсякденному житті.

1 ОГЛЯД ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Огляд предметної області

Месенджер [1] – програмне забезпечення (ПЗ), призначене для миттєвого обміну повідомленнями між користувачами через мережу Інтернет [2]. Основною метою месенджера є забезпечення одночасно зручної, швидкої та безпечної комунікації [3].

Функціонал сучасних месенджерів є неймовірно багатим і постійно розширюється, однак їх базові можливості залишаються незмінними:

- миттєвий обмін текстовими повідомленнями в режимі реального часу;
- передача мультимедійних файлів – фото, відео, аудіо, документи, а також голосові повідомлення;
- групові чати – спілкування в межах спільнот, команд або родин;
- голосові та відеодзвінки, зокрема з підтримкою конференц-режиму;
- підтримка чат-ботів – автоматизованих помічників, які можуть бути як простими програмами, так і інтелектуальними системами на основі ШІ;
- інтеграція з іншими сервісами – не тільки базовими соцмережами, а й календарями, хмарними сховищами, браузерами, зовнішніми сервісами та багато чого, що лише зможе придумати уява розробників.

Крім цього, месенджери все частіше використовують функції персоналізації, реакції на повідомлення, інтеграцію з соціальними мережами, а також інструменти для спільної роботи (наприклад, спільні документи, завдання, опитування) [4]. Месенджери можна класифікувати за різними ознаками: за середовищем запуску застосунку – десктопні, мобільні, веб-версії; за ціновою доступністю: безкоштовні (free), умовно-безкоштовні (freemium), платні (premium); за рівнем безпеки – з шифруванням (p2p, end-to-end) та без шифрування; за призначенням – універсальні, корпоративні або

спеціалізовані. Спеціалізовані месенджери приділяють увагу певній ніші замість масової аудиторії (наприклад, Discord є популярним вибором для зв’язку в ігрових спільнотах).

Можна привести приклади месенджерів для різних цих класифікації. Прикладами універсальних месенджерів для якомога широкої аудиторії є WhatsApp, Telegram та Viber. Прикладами корпоративних рішень є Microsoft Teams, Slack, частково Zoom Chat. Прикладами месенджерів з підвищеним захистом є Signal та Threema [5].

Месенджери зазвичай мають клієнт-серверну архітектуру [6]. Користувачі взаємодіють з інтерфейсом клієнта, який надсилає запити на сервери (вони одночасно є віддаленими для безпеки і найближчими до користувача задля найбільшої швидкості доставки), де обробляються повідомлення, зберігається історія чатів, керуються групами тощо.

Деякі месенджери, як-от Signal, використовують децентралізовану архітектуру або p2p-з'єднання, що підвищує рівень конфіденційності і ще менше знижує шанси на те, що повідомлення буде прочитане третьою стороною.

Однією з ключових вимог до месенджера є безпека. Основні технології безпеки, що активно використовуються в сучасних месенджерах, наступні:

- шифрування (end-to-end encryption) – забезпечує, що повідомлення доступне лише відправнику та отримувачу, і ніяка третя особа не зможе дізнатися про зміст;

- двофакторна аутентифікація – окрім стандартного входу ще додається його підтвердження за допомогою особистих даних користувача, як от електронної пошти, мобільному номеру телефону чи за допомогою механізму всередині застосунку;

- самознищення повідомлень, їх тимчасове надсилання (будь-то на визначений час чи поки розмовник не прочитає зміст);

- захист від фішингу, спаму та вторгнення за допомогою налаштувань приватності для користувачів.

Сьогодні кількість проблем у месенджерах лише росте. При розробці застосунку бажано або вирішити їх, або зробити компромісне рішення. Ключовими та актуальними було виділено наступні:

- захист персональних даних: зростаюче занепокоєння викликає детальний збір і обробка персональної інформації користувачів. Наприклад, Facebook Messenger та WhatsApp часто критикують за політику конфіденційності;

- сумісність і інтероперабельність: відсутність єдиного стандарту перешкоджає взаємодії між месенджерами. Користувачі вимушені встановлювати кілька програм для спілкування з різними контактами;

- надмірна складність: інтерфейс деяких месенджерів перевантажено функціями, що ускладнює їх використання недосвідченими користувачами. Важливо зберігти доступність для користувачів хоча б більшості основних функцій, не навантажуючи інтерфейс.

У майбутньому розвитку месенджерів очікується подальше розширення функціоналу наступними способами:

- автоматичний переклад повідомлень у режимі реального часу (цей фунціонал вже є в premium-версії Telegram);

- інтелектуальні чат-боти [7] – здатні не лише відповідати на питання, але й проводити повноцінну взаємодію (бронювання, консультації, продаж);

- розпізнавання голосу, обличчя та інших об’єктів у мультимедіа;

- підтримка Web3 і блокчейн-технологій – наприклад, месенджери з вбудованими криптогаманцями або на основі токенізованої взаємодії;

- інтеграція ще більшої кількості функції, не пов’язаної з месенджерами напряму;

- підвищена анонімність – можливість спілкування без прив’язки до номера телефону або реального імені, щонайменше, на початкових етапах користування застосунком.

1.2 Аналіз аналогічних застосунків

Серед численних месенджерів багато з них стали особливо популярними завдяки своїм унікальним можливостям. Опишемо кожен з них.

Найпопулярнішим месенджером в світі є WhatsApp [8]. Він забезпечує наскрізне шифрування, а також є неймовірно простим у використанні, причому настільки, що разом з Viber це найкращий вибір для людей похилого віку. У продукті мінімум функціоналу, що схований від очей користувачів, що робить використання месенджером зрозумілим. Але мінуси також будуть: соціальні рішення від Meta часто піддаються масовим зливанням особистих даних користувачів, і WhatsApp не є винятком. Порівняно з іншими месенджерами цей є менш безпечним.

Зараз Telegram (або TG) є вкрай прогресивним рішенням на ринку. За завіреннями розробників новий функціонал додається ледь не щомісяця. З останнього у застосунку переробили систему групових дзвінків, додали подарунки, які можуть слугувати цифровими вкладеннями на кшталт NFT [9], а також додали офіційну підтримку чатів-пропозицій для каналів. Крім того, продукт орієнтований на швидкість і безпеку, наскільки це може робити масовий месенджер.

Але екосистема зі свободою слова, яку збудував застосунок, не завджи є плюсом. Саме в Telegram дуже активно працюють злочинці, які можуть не тільки викрадати особисті дані користувачів (а з нещодавньої пори ще й подарунки), а ще й займатися більш серйозними справами, як от організація терористичних актів або обіг незаконних речовин. Причому знайти такі середовища злочинності відносно нескладно, а іноді вони самі знаходять користувача.

Приклад інтерфейсу застосунку Telegram зображено на рисунку 1.1.

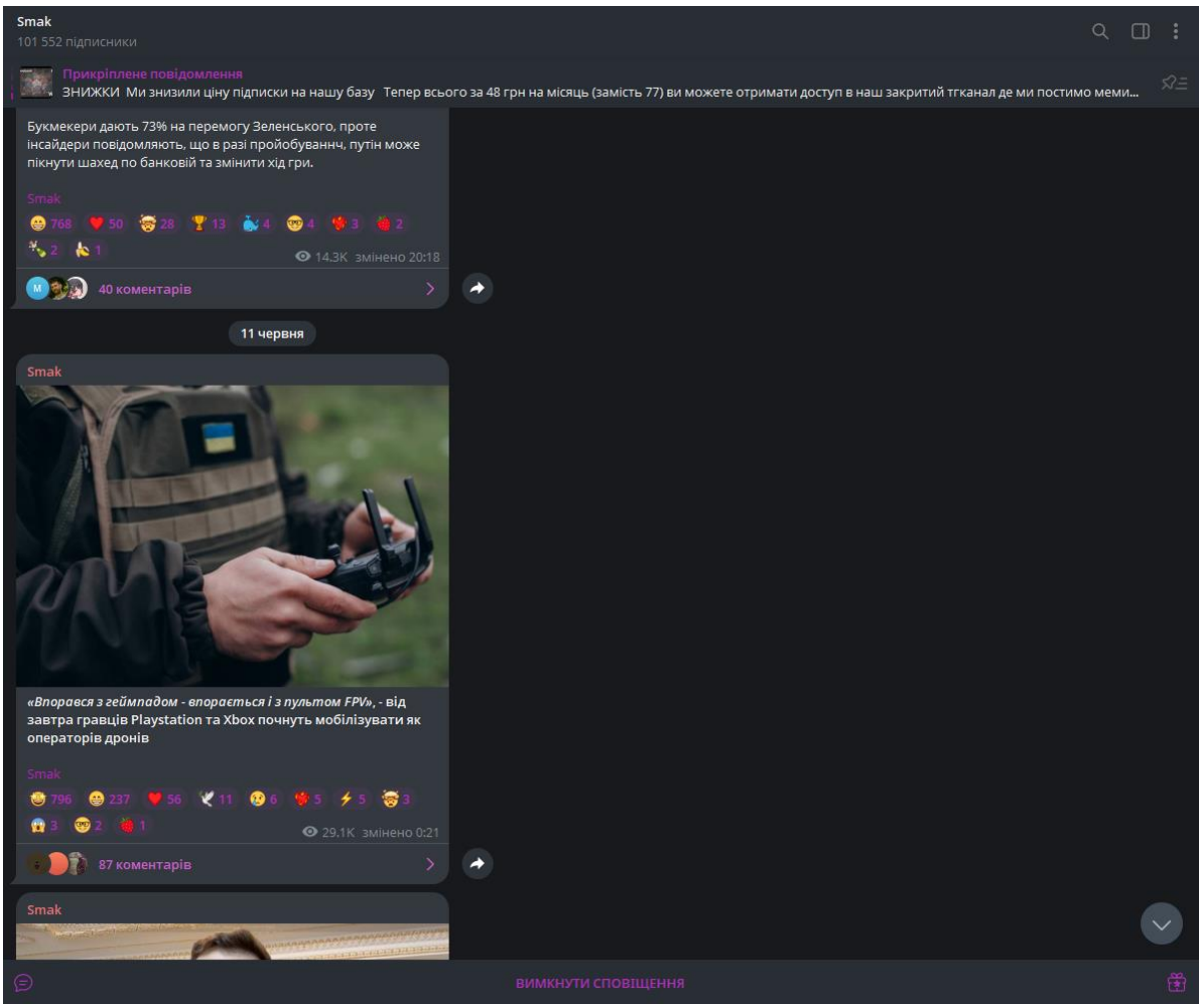

Рисунок 1.1 – Інтерфейс каналу в Telegram

Третім неймовірно популярним масовим месенджером є Viber. Переважно він має ті самі переваги, що і WhatsApp – доступність функціоналу, низький поріг входження, забезпечення базової безпеки. Але до них ще додається грамотна інтеграція більш нішевих функцій: наприклад, реалізація бізнес-чатів у Viber є більш зрозумілою для користувача, аніж у Telegram. Або підтримка дзвінків по стільниковому зв’язку замість Інтернету за допомогою Viber Out. Проте головним мінусом застосунку є те, що у нього немає killer-feature, що змогла би переконати користувачів використовувати Viber замість іншого месенджера. Люди його використовують або з особистої мотивації (наприклад, знайомий або родич є лише там), або разом з іншими.

На рисунку 1.2 зображено інтерфейс чату у Viber.

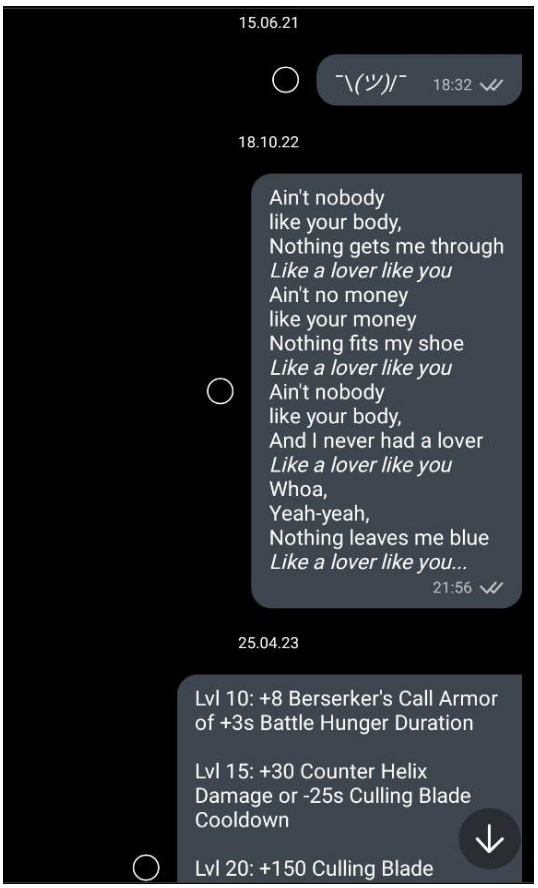

Рисунок 1.2 – Інтерфейс чату в Viber

Зазвичай люди не звилки називати Discord месенджером. Проте нерідко користувачі вирішують використовувати саме його в деяких сценаріях. Його сильними сторонами є висока якість дзвінків, неймовірні можливості для модерації спільнот, а також інтеграції з різними сервісами, причому не лише з цифровими магазинами (Steam, Epic Games Store) і соцмережами (Facebook, Reddit, TikTok), а й з деякими потоковими сервісами, такими як Spotify або Crunchyroll, а також купою інших сервісів різного напряму, в тому числі і для розробників [10]. За функціоналом він конкурує з Telegram, а в плані персоналізації перевершує його. Проте основним недоліком месенджеру є напрям на нішеву цільову аудиторію: застосунок орієнтований переважно на молодих людей, які захоплюються нерозповсюдженими хобі, такими як, наприклад, відеоігри. Користувачам, у яких дозвілля виглядає іншим чином, знайти щось за інтересами буде складніше.

На рисунку 1.3 зображено інтерфейс серверу у Discord.

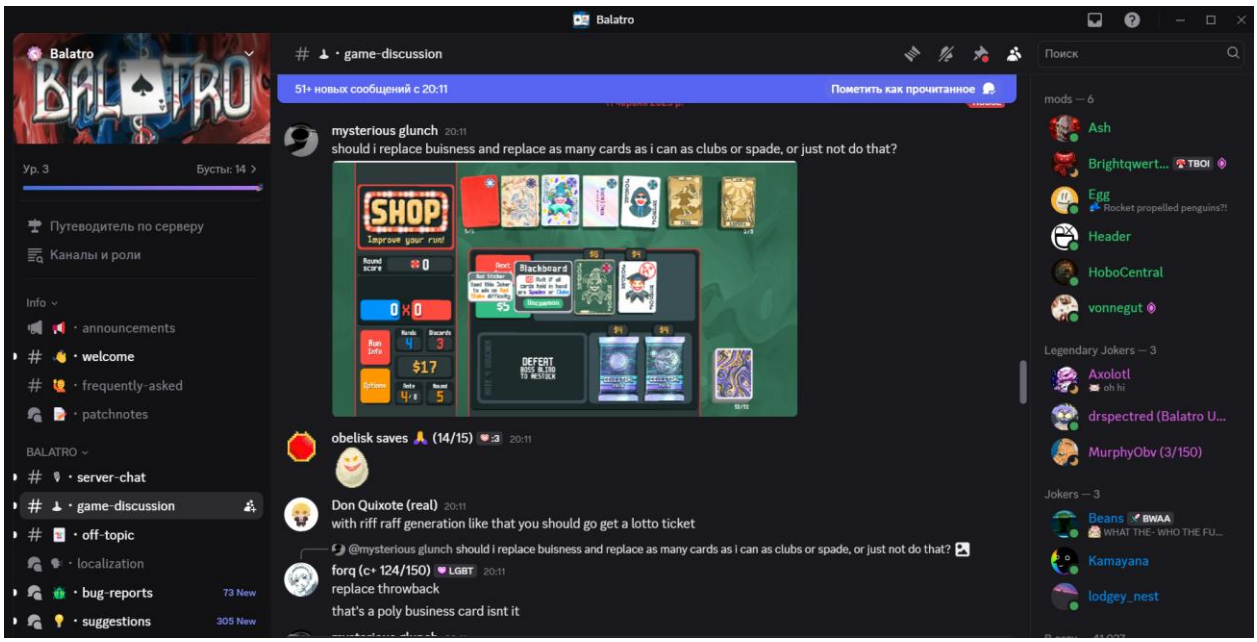

Рисунок 1.3 – Інтерфейс Discord-серверу

Signal є прикладом відносно масового месенджера, орієнтованого не лише на безпеку, а і на приватність користувача. Крім того, це гарантують відкритий вихідний код рішення та шифрування, що працює безпосередньо на пристроях. Цей месенджер є непоганим застосунком, коли мова заходить за по-справжньому безпечне спілкування, але є і недолік, що цей продукт однозначно підходить не для всіх, так як має базову функціональність і не такі великі можливості, які нав’язують гіганти ринку.

Slack та Microsoft Teams є орієнтованими на корпоративне середовище і використовуються в основному для організації робочого простору. Причому для розробників більша перевага надається Slack, оскільки він підтримує безліч інтеграцій зі сторонніми сервісами: від хмарних сховищ Dropbox та Google Drive, сховищ проєктів Github та Bitbucket, до застосунків для планування задач Trello та Jira [11]. А для мінімального наближення до звичайного користувача придатний Teams.

На основі інформації можна скласти певний профіль кожного зі згаданих месенджерів. І на базі цього опису складемо оцінку кожному месенджеру, щоб розуміти, для якої ніші та з якими сильними та слабими сторонами розробляти власний продукт. Оцінка месенджерів є у таблиці 1.1.

Таблиця 1.1 – Оцінка основних месенджерів (максимальний бал - 10)

Критерій	Застосунок						
	WhatsApp	TG	Viber	Discord	Signal	Slack	Teams
Поріг входження	8	7	8	6	6	7	7
Базові функції	8	8	8	8	8	8	8
Додаткові фішки	6	9	7	9	5	8	7
Швидкодія	7	8	7	7	7	7	7
Персоналізація	6	7	6	8	4	5	5
Налаштування	6	8	6	9	6	6	6
Масовість	9	8	7	5	6	7	5
Безпека	6	8	7	7	9	7	7

Після складання оцінки стає зрозумілим, що вирішальну роль у сприйнятті продукту значущу роль відіграє цільова аудиторія застосунку, яку розробники намагаються привабити. Можна піти як шляхом масовості, як от гіганти ринку Viber, Telegram та WhatsApp і розробляти функції для якомога більшого числа людей. Або можна взяти якусь одну характерну перевагу цільової аудиторії, і наголошувати саме на неї, як це роблять Discord, Signal або Slack. Бажана цільова аудиторія стане одним з визначальних факторів при плануванні розробки власного застосунку.

1.3 Аналіз поставленої задачі

Розробка веб-месенджера – це багатогранна інженерна робота, яка вимагає ретельного аналізу, ретельного планування та стратегічного виконання. На самому початку необхідно провести комплексний аналіз вимог та користувачів, враховуючи бачення зацікавлених сторін, функціональні вимоги та очікування користувачів – від базового обміну

миттєвими повідомленнями до розширених функцій, таких як групові чати, обмін файлами, аудіо- та відеодзвінки, тимчасові повідомлення, стикери та підтримка мультимедіа. Аналіз допомагає визначити надійну архітектуру: вибрати відповідний стек (наприклад, Node.js, Python/Django для бекенду; React або Vue для фронтенду), розробити базу даних зі стратегіями шардування або реплікації для повідомлень та медіа, використовувати черги повідомлень (Kafka або RabbitMQ) для надійної, упорядкованої та ідемпотентної доставки, а також використовувати WebSocket або HTTP/2 для двонаправленого зв'язку в режимі реального часу [12]. Забезпечення масштабованості та відмовостійкості є надзвичайно важливим: використання горизонтального масштабування за допомогою груп автоматичного масштабування, балансування навантаження, розподілу CDN для низької затримки та географічного покриття, резервування для рівнів послуг та баз даних, а також планів аварійного відновлення з резервними копіями.

Не менш важливим питанням є безпека та відповідність вимогам, включаючи TLS/SSL для даних під час передачі, шифрування AES-256 у стані спокою, додаткове наскрізне шифрування для забезпечення конфіденційності, автентифікацію через JWT або OAuth, контроль доступу на основі ролей, обмеження швидкості, захист від DDoS-атак, дроселювання IP-адрес та відповідність нормативним вимогам (GDPR, HIPAA).

Окрім основного обміну повідомленнями, аналіз платформи також має підкреслити потребу в насичених компонентах для розмов, таких як мультимедійні вкладення, шаблони, швидкі відповіді, індикатори введення тексту, сповіщення про прочитання, онлайн-присутність, тимчасові «секретні чати», функції скасування відправлення, інструменти модерації каналів та насичені інтерфейси, що поєднують розмовний текст зі структурованими інтерфейсами або вбудованими переглядами, підтримка обміну файлами та медіа також вимагає масштабованого серверного сховища (наприклад, AWS S3, GCP), генерації мініатюр, обробки типів MIME та безпечної передачі та зберігання.

Якщо потрібні аудіо/відеодзвінки, інтеграція з постачальниками послуг, такими як Twilio або WebRTC, є обов'язковою, а також врахування медіасерверів, проходження NAT, адаптації якості та SDK для кожної платформи. Щоб забезпечити безперебійний користувацький досвід та утримання користувачів, необхідно інвестувати в UX/UI дизайн [13], евристичну оцінку, тестування зручності використання та ітеративні цикли вдосконалення, забезпечуючи чіткість, збереження контексту, запобігання помилкам та коректне відновлення після збоїв.

З боку команди та процесу, структурована методологія RMS є важливою: керівник проєкту координує, UX/UI дизайнери створюють прототипи та виконують ітерації, розробники створюють, інженери з контролю якості тестують на кожному етапі, а DevOps забезпечує конвеєри CI/CD, моніторинг, сповіщення та розгортання у виробничому середовищі. Остаточний аналіз також повинен намітити дорожню карту для вдосконалення, включаючи аналітику використання, залученості, затримки повідомлень, рівня помилок та подій безпеки, а також список продуктів для нових функцій, таких як чат-боти, платежі, обмін геолокацією, ігри або екосистеми цифрових наліпок. Крім того, необхідно проаналізувати ризики – від порушень конфіденційності, штрафів за дотримання вимог, низької продуктивності до експлойтів безпеки. Витрати на інфраструктуру, операції, зарплати команди, ліцензування та маркетинг повинні бути оцінені та оптимізовані.

Як висновок, аналітичне завдання розробки веб-месенджера охоплює широкий спектр дисциплін – управління продуктом, UX-дизайн, розподілені системи реального часу, криптографію, масштабовану архітектуру, обробку мультимедіа, відповідність вимогам та ітеративну доставку. Кожен рівень – від початкового збору вимог до архітектури, безпеки, медіа-конвеєрів, розгортання та постійних удосконалень – має бути ретельно спланований, скоординований та виміряний. Тільки завдяки такому цілісному та ітеративному аналізу команда розробників може створити безпечний,

масштабований та орієнтований на користувача веб-месенджер, який процвітає у висококонкурентній цифровій екосистемі.

1.4 Постановка задачі

Головна задача – розробити клієнт-серверний вебзастосунок для спілкування між користувачами, що відповідає основним вимогам, які необхічні не тільки для стабільного функціонування, але і для безпечного використання месенджеру.

Першою обов’язковою фунцкіональною вимогою є реєстрація та автентифікація. Кожен користувач повинен ідентифікуватись у застосунку, і найбільш поширений (і разом з цим найбільш практичний) спосіб це зробити – обліковий запис. На цьому етапі повинна відбуватись реєстрація нового користувача з його валідацією (ім’я, електронна пошта або нікнейм), вхід через пошту/username та пароль. Сесія користувача повинна зберігатись глобально (як на клієнті, так і на сервері), а для облікових засобів необхідно додати персоналізацію.

Другою обов’язковою вимогою є додавання основних можливостей. Месенджером не буде зручно користуватись, якщо в ньому не буде хоча б ключових речей. Позначимо за абсолютний мінімум відображення списку контактів (це або користувачі, чиї дані відомі в обліковому записі, або ті, з ким вже є листування), створення та перегляд чатів, та можливість надсилати текстові повідомлення.

Також буде корисно додати додатковий функціонал, який розкриє можливості основного або допоможе використовувати його правильно. Крупними нововведеннями такого є відправка файлів [14] (зокрема, мультимедіа), система push-сповіщень для миттєвого інформування навіть поза мережею, система статусів користувачів і повідомлень, що оновлюється в реальному часі, а також взаємодію з повідомленнями (в ідеалі взаємодією є видалення, прикріплення, зміна змісту, відповідь на повідомлення, а також

його пересилання).

Але не треба зупинятися лише на великих за розміром функціях. Дрібниці теж грають значущу роль у формуванні користувацького досвіду. Навіть така річ як кнопка скролу чату до найновіших повідомлень є доволі важливою. Тут можна придумати ледь не що завгодно. З популярних у масових месенджерах активно використовують попередній перегляд таких речей, як передача файлів, вищезгадану кнопку прокручування чату, звуки при нових повідомленнях (зокрема у чаті, де вони пишуться), система тек, що дозволяє групувати чати за певною ознакою і т. д.

Також, окрім додавання можливостей, треба забезпечити їх стабільну та безвідмовну роботу, а також практичне, безпечне і стабільне використання. Тому разом з функціональними вимогами формуються і нефункціональні.

Першою такою є безпека. Це просто фундаметальна вимога, без дотримання якої не треба братися за розробку та підтримку месенджера. Не дивлячись на прикладаємі зусилля, наразі кожен застосунок в будь-який момент може зазнати хакерської атаки. Тому безпековою ціллю буде зменшити шкоду від потенційних атак шляхом розробки та використання алгоритмів для захисту даних. Способами досягнути цього є використання HTTPS для запитів [15], шифрування повідомлень при передачі через WebSocket, захист від поширених веб-вразливостей (наприклад, XSS, CSRF), мінімізація доступу до особистих даних користувачів без нагальної потреби.

Іншими способами збільшити вірогідність створення безпечного середовища є грамотне зберігання облікових даних та ключів шифрування. Наприклад, паролі бажано хешувати на сервері: оскільки ця операція одностороння, процедура в рази зменшує шанси хакера несанкціоновано отримати доступ до облікового запису, так як зводить можливість дізнатись пароль лише до методу підбору (його інша назва – brute force). А при розв’язанні криптографічної задачі метод грубої сили працює лише з простими паролями, які входять в список найгірших. Це є основною

причиною, чому поважаючі безпеку користувача компанії просять його придумати пароль з певними критеріями (наприклад, від 8 символів з наявністю хоча б однієї цифри, а також по одному символу верхнього та нижнього регістрів).

Також месенджер повинен бути продуктивним, зокрема при великій кількості користувачів. Затримка доставки повідомлення в межах одного регіону не повинна перевищувати 300 мс. Це гарантує актуальність повідомлень, особливо в сьогодення, коли інформація може стати неактуальною вже через годину. Також важливо ефективно працювати з великими об’ємами повідомлень, особливо коли це стосується медіафайлів. Головними способами обробляти такі дані є кешування, а також оптимізація запитів на сервер та базу даних.

Крім продуктивності повинна бути і масштабованість, оскільки популярні месенджери обслуговують мільйони користувачів. Для ефективної роботи у таких масштабах потрібно функціонально забезпечити підтримку горизонтального масштабування серверів, а також виносити деякі блоки (наприклад, автентифікацію чи зберігання медіа) в окремі сервіси для групування. Це одна з вимог, яка стосується в тому числі і апаратної бази.

Також важливою вимогою є дотримання стандартів UI/UX. Не важливо, скільки корисних функцій зарито в месенджері, якщо користувач не зможе дістатися до жодної з них. Ідеальними кондиціями є сучасний дизайн з фокусом на доступність, швидкість і простоту, адаптивний інтерфейс, який буде підлаштовуватись під інші розміри екрану, а також підтримка світлої і темної теми інтерфейсу – обов’язкової деталі у сучасних застосунках. Крім того, важливо, щоб кожна кнопка, вкладка, вікно та будь-який інший елемент був не лише на своєму місці, а і функціонував так, як на них вказано. Наприклад, працююча кнопка дзвінка абоненту в месенджері не повинна знаходитись у вікні з налаштуваннями і мати піктограму стрілочки вгору. Найбільш вірогідною реакцію користувача на подібні рішення буде спантеличення – він банально не розумітиме, що ця кнопка тут робить.

Крім цих вимог, месенджер повинен бути надійним. Доставка повідомлень повинна відбутись, якщо немає перешкод між користувачами (такою є, наприклад, відсутність підключення до мережі у користувача), а також будь-які елементи, стани та інші деталі не повинні бути непідконтрольними. У користувача буде підірвана довіра, якщо його чат з розмовником сьогодні має один зміст, а завтра – радикально інший. Основними способами вирішення цієї проблеми є резервне копіювання [16], а також синхронізація повідомлень (в тому числі у випадку обриву з'єднання).

Як підсумок, головною метою проєкту є розробка застосунку для спілкування між користувачами. Для того, щоб виконати цю мету, необхідно вирішити наступні задачі:

- розробити процедури реєстрації, авторизації, а також зберігання даних;
- додати можливості для ідентифікації та персоналізації, щоб користувачі могли впізнати одне одного у застосунку;
- додати основний функціонал для спілкування між розмовниками: повідомлення, дзвінки тощо;
- розробити та додати додаткові функції, які спрощують спілкування за допомогою основних: наприклад, відповіді на повідомлення;
- забезпечити стабільну роботу всього функціоналу від реєстрації до видалення повідомлень за допомогою серверу;
- провести тестування застосунку з метою знаходження вузьких місць застосунку, його продуктивності, швидкодії, а також критичних помилок для виправлення.

2 ВИБІР ЗАСОБІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Обґрунтування напряму розробки

Під час розробки месенджеру одним із основоположних рішень є вибір відповідної цільової платформи пристрою. Цей вибір є ключовим, оскільки він впливає на охоплення, продуктивність, користувацький досвід та загальну стратегію розробки додатка. Основні варіанти включають розробку для мобільних пристроїв (iOS та Android), настільних платформ (Windows, macOS, Linux) або використання кросплатформеного підходу, який охоплює кілька типів пристроїв.

Мобільні платформи (iOS та Android) [17]: Орієнтація на мобільні пристрої часто є пріоритетною через повсюдне використання смартфонів для миттєвого зв'язку. Розробка нативних додатків для iOS та Android забезпечує оптимальну продуктивність та доступ до функцій, специфічних для пристрою. Нативна розробка використовує мови та інструменти, специфічні для платформи – Swift або Objective-C для iOS та Kotlin або Java для Android – забезпечуючи безперебійний користувацький досвід. Однак цей підхід вимагає підтримки окремих кодових баз для кожної платформи, що потенційно збільшує час та ресурси розробки.

Настільні платформи (Windows, macOS, Linux): Хоча використання мобільних пристроїв домінує, настільні платформи залишаються актуальними, особливо в професійному або корпоративному середовищі, де користувачі можуть віддавати перевагу або потребувати настільних додатків. Розробка для настільних комп'ютерів може покращити функції продуктивності, такі як інтеграція з іншими настільними програмами та підтримка більших екранів. Однак різноманітність настільних операційних систем та порівняно менша база користувачів можуть створювати проблеми з точки зору складності розробки та розподілу ресурсів.

Кросплатформена розробка: Щоб пом'якшити проблеми підтримки кількох кодових баз, популярність набули фреймворки для кросплатформеної розробки, такі як React Native або Flutter [18]. Ці фреймворки дозволяють розробникам писати єдину кодову базу, яку можна розгорнути на кількох платформах, включаючи iOS, Android і навіть веб- або настільні програми. Такий підхід може значно скоротити час розробки та витрати, забезпечуючи при цьому однаковий користувацький досвід на всіх пристроях. Однак він може мати компроміси з точки зору продуктивності та доступу до певних нативних функцій.

Стратегічні міркування: Рішення щодо цільового пристрою має відповідати цільовій аудиторії програми, варіантам використання та бізнес-цілям. Наприклад, якщо цільова аудиторія переважно використовує мобільні пристрої, доцільно зосередитися на мобільних платформах. І навпаки, якщо програма призначена для професійного використання, включення підтримки настільних комп'ютерів може бути важливим. Крім того, врахування таких факторів, як ресурси для розробки, час виходу на ринок та довгострокове обслуговування, має вирішальне значення для вибору найбільш підходящого напрямку розвитку.

Окрім технічних міркувань щодо цільових пристроїв, вкрай важливо узгодити напрямок розробки з основними цілями месенджеру. Цих цілей безліч, але основними з них є охоплення зв'язку в реальному часі, конфіденційність користувачів, безперебійний обмін медіафайлами та широка доступність. Кожна з цих цілей накладає певні вимоги та впливає на вибір стратегій та технологій розробки. Про них ми детально і поговоримо.

Зв'язок у реальному часі [19]: в основі будь-якого месенджеру лежить здатність сприяти зв'язку в реальному часі. Досягнення доставки повідомлень з мінімальною затримкою та миттєвої синхронізації між пристроями вимагає надійної інфраструктури серверної частини та ефективної обробки даних. Такі технології, як WebSocket та WebRTC, відіграють важливу роль у забезпеченні постійних двонаправлених каналів зв'язку між клієнтами та

серверами. Вибір платформи може вплинути на впровадження цих технологій; наприклад, мобільні платформи можуть мати обмеження у підтримці постійних з'єднань через політики управління живленням, тоді як настільні платформи можуть пропонувати більшу стабільність у цьому відношенні.

Конфіденційність та безпека користувачів: Забезпечення конфіденційності користувачів має першорядне значення в месенджер-додатках. Впровадження наскрізного шифрування є критично важливим заходом для захисту вмісту повідомлень від несанкціонованого доступу. Такі фреймворки, як Signal Protocol, забезпечують надійні механізми шифрування, які можна інтегрувати в месенджер-платформи. Однак, інтеграція E2EE створює складнощі в синхронізації та зберіганні повідомлень, особливо коли користувачі отримують доступ до повідомлень на кількох пристроях. Розробники повинні розробляти безпечні системи керування ключами та враховувати наслідки зберігання зашифрованих повідомлень на серверах, гарантуючи, що лише авторизовані пристрої можуть розшифрувати вміст.

Безперешкодний обмін медіа: Сучасні користувачі очікують, що програми обміну повідомленнями підтримуватимуть обмін різними типами медіа, включаючи зображення, відео, документи та голосові повідомлення. Забезпечення безперешкодного обміну медіа вимагає ефективної обробки завантажень та завантажень файлів, стиснення медіа, його кешування та сумісності з різними форматами файлів [20]. Напрямок розробки повинен враховувати можливості цільових пристроїв; наприклад, мобільні пристрої можуть мати обмеження в обробці великих медіафайлів, що вимагає стратегій оптимізації для забезпечення безперебійної роботи.

Широка доступність: щоб охопити різноманітну базу користувачів, програми обміну повідомленнями повинні бути доступні на різних пристроях та платформах. Застосування кросплатформеного підходу до розробки може покращити доступність, забезпечуючи узгоджений користувацький досвід на різних пристроях. Однак розробники повинні забезпечити, щоб програма

відповідала стандартам доступності, таким як підтримка програм зчитування з екрана та врахування потреб користувачів з інвалідністю. Крім того, для задоволення потреб користувачів з різних куточків світу потрібно підтримувати локалізації застосунку. Тобто месенджеру бажано підтримувати кілька мов і дотримуватися їх правил (буде відверто поганим рішенням додати в застосунок арабську локалізацію, але з написанням слів зліва направо), а також культурних нюансів, що притаманні жителям регіонів, де розгортається месенджер.

Масштабованість та продуктивність: зі зростанням бази користувачів месенджера система повинна масштабуватися, щоб обробляти збільшений трафік та підтримувати продуктивність. Це вимагає масштабованої архітектури серверної частини, використовуючи такі методи, як балансування навантаження, шардування бази даних, мікросервіси, а також актуальної апаратної бази для обробки цих даних (сервери також активно розвиваються з точки зору продуктивності). Вибір напрямку розробки повинен враховувати вимоги масштабованості, гарантуючи, що вибрані технології та платформи можуть забезпечити майбутнє зростання без шкоди для продуктивності.

Відповідність нормативним вимогам: месенджери повинні відповідати різним нормам захисту даних та конфіденційності. Прикладом такого нормативу є Загальний Регламент з Захисту Даних (GDPR) у Європейському Союзі. Вимоги до відповідності впливають на рішення щодо розробки, включаючи практику зберігання даних, механізми згоди користувачів на обробку персональної інформації та саму обробку у застосунку. Розробники повинні забезпечити відповідність архітектури та процесів обробки даних месенджера нормативним стандартам, які можуть відрізнятися залежно від регіонів, де месенджер розгортатиметься.

2.2 Обґрунтування стеку програмування

Тепер коли визначились з основними положеннями для обґрунтування напрямку програмування, настав час визначитись зі стеком. Переважно він залежить від кінцевого пристрою, але в ньому враховується специфіка платформи і бачення розробника.

Цільовою платформою для застосунку буде браузер. Причиною цього вибору стала потенційна можливість охопити одразу десктопні і мобільні пристрої одним рішенням. Єдиний нюанс – для повного використання цієї переваги необхідно навчитися адаптивно верстати структуру документу та стилізувати його. В усіх браузерних застосунках основа одна: HTML [21].

HTML – фундамент будь-якого сайту або вебзастосунку. Майже всі рішення в тому чи іншому вигляді використовують HTML як засіб для розмітки змісту. Він одночасно і простий у засвоєнні, і легко читається розробником і має доволі багато функціоналу. Однозначний фаворит для основи застосунку. У лістингу 2.1 зображено розмітку для вікна логіну, а в його коментарях призначення використаних тегів

Лістинг 2.1 – Приклад вікна логіну у HTML-розмітці

```
return (
    <div className="login-container"> // Універсальний блочний
тег div
      <h2>Login</h2> // Тег заголовку 2-го рівня
      <input // Тег поля введення
        type="text"
        placeholder="Username" // Текст-плейсхолер, що
показується до введення користувачем
        value={username} // Значення поля, яке згодом
передасться у функцію
        onChange={(e) => setUsername(e.target.value)}
      /> // Закриття тегу (цей синтаксис прийнятний у React)
      <input
        type="password" // Тип зображення тексту у полі. Тут це
пароль
        placeholder="Password"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
      />
```

```
      <button onClick={handleLogin}>Login</button> // Кнопка.
При натисненні на неї спрацьовує івент onClick
      <p>{message}</p> // Текстове поле тегу paragraph
      <p> Do not have an account? <button>Register</button>
      </p> // Закриття тегу у HTML
    </div>
  );
```

Далі вже йдуть насправді необов’язкові елементи. Справа в тому, що в певній мірі застосунок функціонуватиме лише з використанням HTML. Але ця функціональність настільки обмежена, що така структура майже ніколи не використовується на практиці.

Перший з таких елементів – стилізація HTML-документу. Для цього буде використовуватись класичний CSS. Він, як і сам HTML, вже давно перевірений часом і має достатній функціонал для розробки застосунку такого рівня. Гідними для згадки також є препроцесори стилів (наприклад, SASS/SCSS) та фреймворки для CSS (Bootstrap, Tailwind), які також активно використовують для розробки сучасних застосунків. Але основною причиною вибору є сумісність – класичний CSS підтримуються майже усіма браузерами, навіть такими як Internet Explorer, в той час як, наприклад, Bootstrap не підтримує IE в принципі.

Як і було раніше зазначено, CSS не є обов’язковим елементом для застосунку. Але без нього, вигляд сайту нагадуватиме 1990-ті роки, коли пріоритетом було в принципі завантажити сторінку, а не зробити її вигляд привабливою. На рисунку 2.1 зображено вигляд сторінки з використанням CSS, на рисунку 2.2 – без нього.

Рисунок 2.1 – Вигляд HTML-документу з використанням CSS

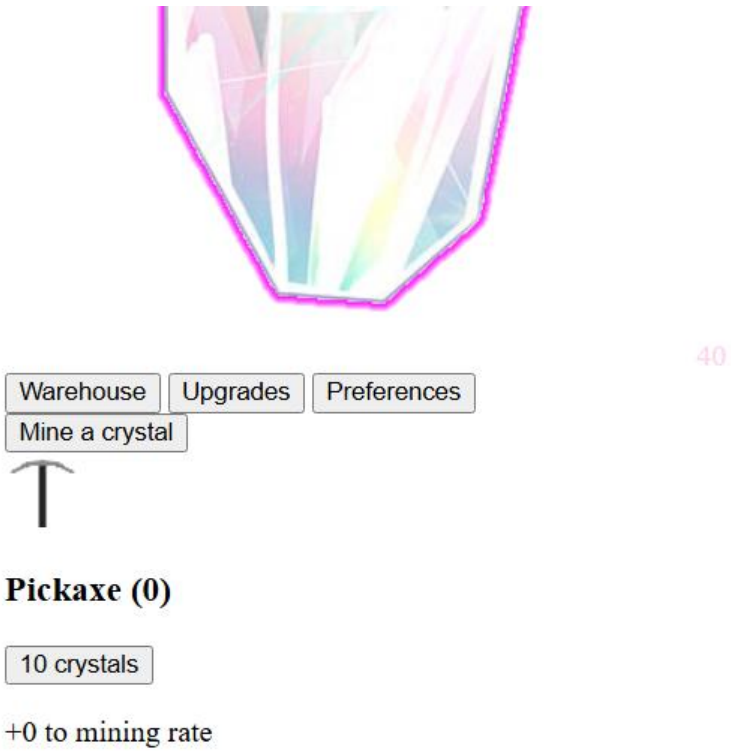

Рисунок 2.2 – Вигляд того ж самого HTML-документу без використання CSS

Далі у нас JavaScript [22] – основна мова програмування у вебзастосунках. Стала вона основною, оскільки найкраще за всіх використовує основну модель застосунків з точки зору розробки – DOM-дерево. Вона теж номінально є необов'язковою для використання, але застосунок багато без неї втрачає, а саме майже всю функціональну базу. За допомогою HTML + CSS можна зробити зовнішній вигляд проєкту. Але ці мови жодним чином не допомагають у тому, щоб цей зовнішній вигляд реагував на дії користувача.

Після вибору мови програмування ми стоїмо на роздоріжжі: класичний JavaScript чудово виконує поставлені для нього задачі, але у нього є одна велика проблема – він не дозволяє зробити вебзастосунок односторінковим [23]. Будь-який великий сайт буде витрачати час на завантаження додаткових HTML-сторінок або додаткового змісту поточної сторінки, і не факт, що абсолютно весь зміст знадобиться одразу після запуску. Наприклад, завантажувати у HTML весь функціонал застосунку разом з вікном реєстрації просто недоречно – користувач не зможе ним скористатись.

Для месенджера, де швидкість взаємодії відіграє велику роль, це є сильним ударом. Якщо використовувати в таких проєктах звичайний JavaScript або його типізовану версію, TypeScript, то необхідно жертвувати або швидкодією, або гнучкістю, або функціональністю. А нерідко при розробці не вдається зберегти одразу всі ці переваги.

Тому було знайдено елегантне рішення – фреймворки. Для вебзастосунків існує три популярних фреймворки – Angular, Vue.js та ReactJS. Вони виконують приблизно однакову основну роль, але у них різне призначення. Поговоримо про них по порядку.

Почнемо з Angular [24]. За можливостями він найбільш обширний, але одночасно з цим найбільш неповороткий з фреймворків. Він має двосторонню прив’язку даних за замовчуванням, що спрощує роботу для сервера, і цільову підтримку архітектурних патернів MVC та MVVM. Це робить Angular ідеальним для використання в розробці рішень для бізнесу та enterprise-сектору. Проте у нього є і солідні мінуси – про нього не можна сказати, що він гнучкий та продуктивний, зокрема у великих застосунках. З точки зору розробника недоліком є те, що для цього фреймворку складніше писати код порівняно з іншими. Також мінусом є те, що нові версії Angular підтримують лише TypeScript.

Vue.js [25] навпаки є найбільш легким фреймворком з трьох. Розробник може легко навчитися йому, а сам він надає обширний набір функцій, який мінімально впливає на продуктивність. Достатньо використовувати HTML + JavaScript, але якщо виникне потреба, то можна використовувати і TypeScript. Застосунки на Vue зазвичай не є великими, адже з цим пов’язаний головний мінус – обмежена гнучкість та підвищена складність масштабованості. Розробники фреймворку спробували вирішити проблему шляхом використання модульності та додавання можливості використання компонентів, але це у випадку використання у застосунку скоріше за все стане лише можливістю подовжити життєвий цикл підтримки продукту, а не повноцінним рішенням проблеми.

ReactJS [26] же є еталоном універсальності та балансу серед фреймворків. Він підходить як для малих проєктів, так і для великих. На ньому не настільки складно навчитись писати, він використовує вдалу для розробки компонентну модель, а використання хуків дозволяє писати для компонентів гнучку логіку. Він, як і Angular, дозволяє контролювати архітектуру, і такий же високопродуктивний, як і Vue. Але і мінуси у React доволі значущі – з трьох фреймворків цей найменш самостійний, і для доступності деяких функцій до нього обов'язково треба додавати надбудови. Наприклад, у великих застосунках для оптимізації навантаження на кінцевий пристрій при відрисуванні елементів, використовується фреймворк Next.js.

Після опису всіх фреймворків можна скласти таблицю 2.1 з оцінкою «підходящості» фрейморку для розробки цільового застосунку. Всі оцінки використовують 10-бальну шкалу: в усіх критеріях більша цифра означає кращий результат, за винятком впливу недоліків не розробку: там найкращому результату відповідає менша цифра.

Таблиця 2.1 – Приблизна оцінка фреймворків для JavaScript за критеріями

Критерій	Фреймворк		
	Angular	React	Vue
Простота навчання	4	6	8
Функціонал для розробки застосунку	8	7	7
Потенціал для масштабованості	9	8	6
Продуктивність	6	8	9
Гнучкість та модульність	6	8	6
Самостійність	7	5	7
Взаємодія фреймворку з сервером	8	7	7
Вплив недоліків на розробку	7	4	5

Після всіх зважувань щодо фреймворку, вибір пав на React, як на універсальне рішення, що стане в нагоді не тільки на ранніх етапах, як Vue,

але і у випадку, коли буде потрібно масштабувати застосунок. А згодом можна буде додавати надбудови, коли проблеми, які вони вирішують, заважатимуть розвитку функціоналу.

Vue теж був непоганим варіантом для вибору, але нестача потенціалу для масштабованості застосунку є занадто великим недоліком. Плюс навіть якщо спробувати піти в цьому напряму, фреймворк недостатньо гнучкий. Тоді як Angular не став вибором, оскільки не має високої продуктивності (що для месенджера є критичною проблемою), а також у нього немає гнучкості для застосунків середнього розміру.

Для сервера теж є небагато варіантів, а точніше лише два. Перший – надійний і стабільний Node.js [27]. Другий – лише починаючий підкорювати сьогоднішній ринок Deno. Вони мають схожу структуру, схожий принцип роботи, і створені одним і тим же автором. Використовують для синтаксису одну і ту ж основну мову – JavaScript, а також підтримують більшість одних і тих самих бібліотек. Сервер, що відповідає вимогам, можна написати на обох цих платформах.

Відмінностей не так багато: основними є те, що Deno використовує власний пакетний менеджер, підтримує TypeScript і краще використовує новітні синтаксиси JavaScript, такі як ES6. Також через те, що Deno був написаний за допомогою мови програмування Rust, він буде дещо безпечнішим на етапі компіляції та переведення в машинний код, залишаючись таким же швидким. Для порівняння, Node.js написаний мовами C/C++, які такі ж швидкі, як і Rust, але в яких недосвідченому розробнику достатньо легко добитися витоку пам’яті у програмі чи функції. Цей факт зазвичай і робить різницю в безпеці компіляції.

В проєкті буде використовуватись Node.js, оскільки відмінності Deno, які повинні робити його краще, не впливають на роботу – він буде писатись на JavaScript, а не TypeScript. Менеджер пакетів NPM, що входить в Node.js, є найпопулярнішим у цьому сегменті, що гарантує величезний обсяг бібліотек для використання. Обсяг настільки великий, що іноді можна буде обирати з

декількох способів вирішення проблеми. А грамотна реалізація серверу дозволить нівелювати проблему безпечного переведення в машинний код.

Менеджер пакетів у застосунках використовується для завантаження бібліотек, встановлення залежностей, а також керуванням цих речей у застосунку. Він допоможе миттєво додати у проєкт будь-який необхідний розробнику компонент, а також зібрати всі залежності воєдино.

Для розробки менеджером пакетів обрано NPM, оскільки він інтегрований в уже використовуваний Node.js. NPM, як і всі основні менеджери пакетів, складається з трьох компонентів: сховища, де і перебувають залежності, CLI для контролю розробки з терміналу чи командного рядку, а також регістру, де є мета-інформація про ПЗ, пов’язане з JavaScript. Тобто для пошуку рішення потрібно:

- знайти бібліотеку чи залежність у сховищі (наприклад, для NPM це вебсайт);

- дізнатися про її основні можливості: спільнота, яка активно розроблює такі залежності, зазвичай вказує функціонал для того, щоб інші розробники знали, чи потрібно використовувати ту чи іншу бібліотеку в їх випадку;

- якщо залежність буде корисною, то її можна встановити одразу в застосунок за допомогою команди в терміналі, яка додатково ще буде зазначена в сховищі. Наприклад, щоб встановити ключову для серверу бібліотеку express, достатньо ввести команду npm install express – і майже одразу розробник зможе працювати з залежністю у своєму проєкті.

3 АРХІТЕКТУРА, СТРУКТУРА І ОПИС ПРОГРАМИ

3.1 Архітектура і структура застосунку

Застосунок побудований за принципами клієнт-серверної архітектури, яка дозволяє розділити відповідальність між інтерфейсом користувача (frontend) та логікою обробки даних (backend). Такий підхід є зручним і гнучким для реалізації масштабованого вебзастосунку.

Клієнт написаний на React, що дозволяє створювати динамічний, швидкий, адаптивний та функціональний інтерфейс. Його структура включає такі компоненти:

- UI-компоненти: верстка виконується за допомогою HTML, а стилізація здійснюється за допомогою класичного CSS та бібліотек інтерфейсу. Кожен інтерфейсний елемент (наприклад, повідомлення, вхідна форма, список чатів) реалізований у вигляді окремого компонента, що має свої стилі, які відрізняються одне від одного;

- управління станом [28] (state management): використовується Context API для передачі глобального стану між компонентами (прикладами таких станів є інформація про поточного користувача, активний чат, наявність нових повідомлень). Сам Context API складається з багатьох вкладених в JavaScript та React інструментів, таких як хуки різних типів, а також синхронні та асинхронні функції;

- WebSocket-клієнт: зв’язок із сервером у реальному часі реалізовано за допомогою бібліотеки socket.io-client. Це дозволяє миттєво обмінюватися повідомленнями між клієнтами і оновлювати їх стан;

- валідація форм: для реєстрації, входу та відправки повідомлень застосовується React Hook Form, що дозволяє ефективно контролювати форму та валідацію введених даних. Це зменшує кількість роботи на стадії тестування;

- routing: реалізовано через React Router, що дозволяє здійснювати навігацію між сторінками/екранами застосунку без перезавантаження всієї сторінки.

Бекенд побудований на Node.js, що забезпечує обробку HTTP-запитів, автентифікацію, авторизацію та зв’язок із клієнтом у реальному часі.

Основними елементами серверної частини є:

- express.js: базовий фреймворк для обробки маршрутизації, REST-запитів і middleware-функцій;

- CORS: конфігурація доступу між доменами, необхідна для безпечної взаємодії між клієнтом і сервером;

- WebSocket-сервер: реалізований за допомогою socket.io, відповідає за двосторонню комунікацію між клієнтами;

- база даних: для спрощення MVP використовується зберігання даних у вигляді JSON-файлів. Це дозволяє швидко створити прототип застосунку без налаштування повноцінної СУБД;

- безпека: є комплексною і використовує декілька бібліотек – bcrypt для хешування паролів, jsonwebtoken (JWT) для створення токенів автентифікації та uuidv4 для унікальної ідентифікації чатів;

- файлова ієрархія: збереження завантажених медіафайлів реалізовано через fs та бібліотеку multer, яка дозволяє обробляти multipart/form-data-запити;

- зв’язок: для дзвінків між користувачами використовується технологія зв’язку WebRTC та технологія з’єднання P2P. Вони вже встигли довести свою ефективність на практиці.

На рисунку 3.1 зображено схему, яка описує вплив складових у застосунку.

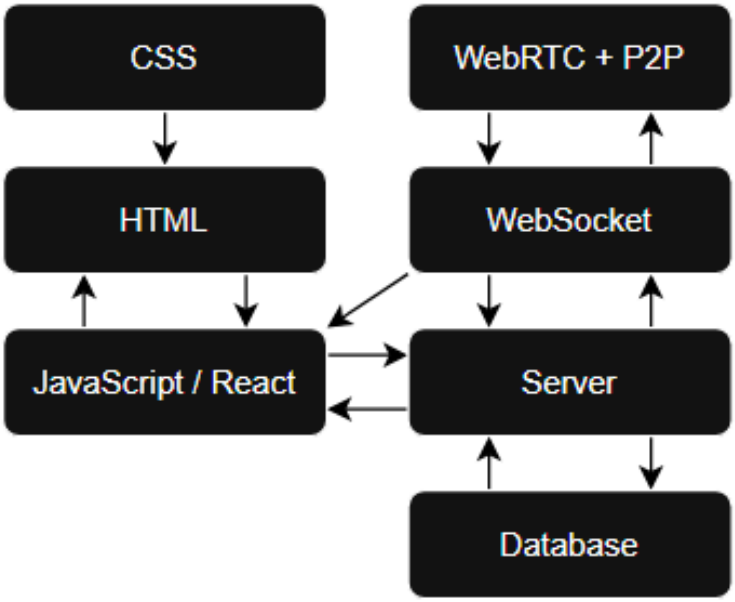

Рисунок 3.1 – Схема впливу компонентів у застосунку

Коли ознайомились з загальною структурою, настав час окреслити, чим саме буде описана функціональність застосунку.

Сервер буде написаний повністю у функціональній парадигмі [29], де одна функція буде відведена під один ендпоінт. Ендпоінти матимуть ієрархічну структуру, тобто вони матимуть шлях, так само, як і директорії в операційних системах (наприклад, шлях для входу буде server/api/login). Потрібно виділити маршрути під вхід, реєстрацію, завантаження даних користувача, зміну цих даних, взаємодію з чатами (наприклад, їх додавання та завантаження їх змісту) та повідомленнями (надсилання, прикріплення, видалення, відповідь на них), на завантаження та збереження файлів.

Клієнт буде написаний зі специфікою фреймворку у функціональній парадигмі і матиме компонентний підхід. У React потрібно підтверджувати передаваємі у компонент поля за допомогою пропсів, а також бажано повертати HTML-компонент. Одним компонентом вважатиметься одне вікно чи його частина: таким буде вважатись, наприклад, модальне вікно логіну або реєстрації. Також компонентом буде інтерфейс чату, дзвінку, список чатів або його окремий елемент.

3.2 Опис програми: фундаментальні функції

Для спрощення написання та збільшення ефективності застосунку [30] потрібно використовувати бібліотеки. Використані для серверу (лістинг 3.1) бібліотеки наступні.

Лістинг 3.1 – Опис бібліотек, використаних для серверу

```
/* Серверна частина: */
import express from 'express'; // Фундаментальна бібліотека для
створення серверу, завдяки якій можливо створювати ендпоінти
import cors from 'cors'; // Бібліотека для підтримки сервером
CORS-протоколів. Вони забезпечують пересилання на сервер лише
безпечних запитів
import fs from 'fs'; // Бібліотека для обробки файлів
import path from 'path'; // Бібліотека для представлення шляхів
import { fileURLToPath } from 'url'; // Функція для отримання
метарозташування до файлу або директорії
import WebSocket, { WebSocketServer } from 'ws'; // Бібліотека
для створення WebSocket-з’єднань
import multer from ‘multer’; // Бібліотека для завантаження
файлів на сервер
import { Buffer } from 'buffer'; // Бібліотека для створення
буферів на сервері.
Import iconv from 'iconv-lite'; // Бібліотека для перетворення
кодування тексту. Разом з 'buffer' допомагає при передачі файлу
зберегти його ім’я
Import {uuidv4} from ‘uuid’ // Бібліотека для створення
універсальних унікальних ID
```

В клієнтській частині бібліотек не так багато, але вони також є необхідними для стабільної роботи. У лістингу 3.2 описані використані залежності.

Лістинг 3.2 – Опис бібліотек, використаних для клієнту

```
/* Клієнтська частина: */
import { useState, useEffect, useRef } from 'react'; // Імпорт
методів з фреймворку React (вони називаються хуками).
import PropTypes from 'prop-types'; // Бібліотека для
представлення компоненту з властивостями
import axios from 'axios'; // Бібліотека для запитів з клієнту
на сервер
import { formatDistanceToNow, format, isSameDay, parseISO } from
'date-fns'; // Бібліотека, що призначена для форматування часу.
```

```
Використовується, наприклад, для розділення повідомлень в чаті
датами їх створення
import ReactMarkdown from 'react-markdown'; // Бібліотека для
підтримки Markdown-стилізації в повідомленнях
import rehypeSanitize from 'rehype-sanitize'; // Плагін для
перевірки безпечності HTML-коду. Санітаризація потрібна для
коректного відображення Markdown-стилізації
```

Після вікна реєстрації та входу користувач потрапляє у основне вікно застосунку. Для того, щоб користувач зміг самовиразитись та дати знати іншим, хто він є, були створені профілі користувача:

- при реєстрації серверний ендпоінт вносить дані до профілю: ім’я для відображення та статус, які є плейсхолдером, а також ім’я користувача, яке вказується при реєстрації;
- у модальному вікні профіля, користувач може змінити свої ім’я для відображення, ім’я користувача (зміна цього пункту тягне за собою введення нового імені користувача при вході в застосунок), а також статус;
- при підтвердженні відбувається запит на сервер, який поверне успіх, якщо нові дані не конфліктують з іншими (наприклад, не вказано вже зайняте ім’я користувача).

Ось так виглядає структура модального вікна профілю (лістинг 3.3).

Лістинг 3.3 – HTML-структура модального вікна профілю

```
return (
    <div className="modal-overlay">
      <div className="modal-content profile-modal">
        <div className="header">
          <h2>Edit Profile</h2>
          <button onClick={onClose} className="close-
button">✕</button>
        </div>
        <form onSubmit={handleSubmit}>
        <label>Avatar:
        <div className="avatar-picker">
            {avatarUrl
              ? <img src={avatarUrl} alt="Avatar Preview"
className="avatar-preview" />
              : <div className="avatar-placeholder">No
Avatar</div>
            }
```

```
            <input
              type="file" accept="image/*" ref={fileInputRef}
              style={{ display: 'none' }}
onChange={handleAvatarSelect}
            />
            <button type="button" onClick={handleDeleteAvatar}>
                Delete Avatar
            </button>
          </div>
          </label>
          <label>Display Name:
            <input type="text" value={displayName} onChange={(e)
=> setDisplayName(e.target.value)} />
          </label>
          <label>Username:
            <input type="text" value={username} onChange={(e) =>
setUsername(e.target.value)} />
          </label>
          <label className="bioLabel">Bio:
            <small>{remainingChars}</small>
          </label>
          <textarea value={bio} onChange={handleBioChange}
maxLength={150} />
          <div className="center-button">
            <button type="submit">Save</button>
          </div>
        </form>
        {message && <p>{message}</p>}
      </div>
    </div>
  );
```

Коли користувач оновив свій профіль, потрібно налаштувати застосунок під себе. Для цього були створені налаштування. Для проєкту було розроблено один глобальний пункт налаштування – тема застосунку. Реалізація цієї функції така.

Спочатку у CSS за допомогою змінних визначається палітра кольорів елементів при темній та світлій темі (в випадку застосунку достатньо кольорів для фону, тексту, та кнопок). Потім у HTML у елементів прописується клас, додавання якого змінює кольори на ті, що вказані у відповідній темі. Далі прописується скрипт, що змінює тему в залежності від вибору пункту – темна чи світла.

Для «системної» теми, яка повторює тему пристрою, на якому застосунок запущений, використовуються медіазапит «prefers-color-scheme»,

що зчитує поточну тему пристрою, на якому застосунок використовується.

Після налаштування можна спробувати і написати комусь повідомлення. Але спочатку потрібно встановити контакт між двома користувачами та створити чат. Для цього був створений відповідний алгоритм.

Додавання контакту починається з натискання кнопки «Contact». Вона створює модальне вікно. Застосунок очікує введення імені користувача (не плутати з ім’ям для відображення displayName), з яким потрібно встановити контакт. При натисканні кнопки підтвердження відправляється запит на сервер.

Якщо сервер знаходить користувача, то створюється чат. Відбувається як серверна ініціалізація, так і створення відповідного елементу у клієнті. Про елемент поговоримо трохи згодом.

У лістингу 3.4 написаний код функції додавання чату до списку.

Лістинг 3.4 – Клієнтська функція додавання чату до списку

```
const handleAddChatSubmit = async (e) => {
    e.preventDefault();
    if (!newUsername.trim()) {
      setErrorMessage('Username cannot be empty');
      return;
    }
    try {
    const response = await
axios.post('http://localhost:3001/addChat', {
      username: newUsername, currentUserId,});
      const newChat = response.data;
      const userResponse = await
axios.get(`http://localhost:3001/ user/${newChat.user2}`);
      const { displayName, avatar } = userResponse.data;
      const updatedChat = { ...newChat, displayName, avatar };
      setChats((prevChats) => [...prevChats, updatedChat]);
      onAddChat(updatedChat);handleCloseModal();
    } catch {setErrorMessage('Failed to add chat. Please try
again.');
}}
```

Після того, як був доданий чат, можна натиснути на нього, аби відкрити інтерфейс чату. Але перед цим поговоримо про сам елемент чату.

Він складається з багатьох інформативних частин, кожна з яких дає користувачу всю наявну інформацію про цей чат.

Спочатку треба сформувати зміст елементу. Для цього обробляються дані про користувача, а також останнє повідомлення. Приклад обробки можна побачити у лістингу 3.5.

Лістинг 3.5 – Форматування елементу чату

```
const { lastMessage: lm, unreadCount = 0 } = chat;

  // Форматування часу
const time = lm ? new Date(lm.timestamp).toLocaleTimeString([],
{ hour:'2-digit', minute:'2-digit' }) : '';
  // Форматування текстового повідомлення
  let preview = 'No messages yet';
  if (lm) {
    if (lm.files?.length) {
      const ft = lm.files[0].fileType || '';
      preview = ft.startsWith('img/') ? 'Photo' :
ft.startsWith('vid/') ? 'Video' : ft.startsWith('audio/') ?
'Audio' : 'File';
    } else {
      preview = lm.content.replace(/[#_*~`>!-]+/g,'').trim();
    }
  }
  // Форматування статусу повідомлення
  let statusIcon = '';
  if (lm?.senderId === currentUserId) {
    statusIcon = lm.read ? '✔✔' : lm.delivered ? '✔' : '';
  }
```

Після формування повідомлення залишилося відобразити все це у елементі в рамках HTML.

Тепер можна поговорити про інтерфейс чату. У ньому багато чого є, тому розповідь йтиме по порядку.

У верхній частині вікна знаходиться інформація про співрозмовника. У ній все доволі просто: відображається основна інформацію зверху, кнопки дзвінка та закріплених повідомлень, а також за умови натискання лівою клавішею миші на ліву частину з’явиться модальне вікно з детальними вписаними розмовником даними. Це виглядає як у лістингу 3.6.

Лістинг 3.6 – HTML-стилізація верхньої частини інтерфейсу чату

```
<div className="chat-header">
            <div className="chat-title"
onClick={openProfileModal}>
              <img
               src={chat.avatar || '/default-avatar.png'}
               alt="Avatar"
               className="chat-avatar" />
              <div className="chatter-text">
                <span className="chatter-name">
{selectedChat.displayName}</span>
                <div className="chat-status">{statusText}</div>
              </div>
            </div>
            <div className="chat-buttons">
            {pinned.length > 0 && (
              <button
                className="pinned-button"
                onClick={() => setShowPinnedModal(true)}
                title="Show pinned"
              >
                📌 {pinned.length}
              </button>
            )}
            <button
            className="call-button"
            onClick={() => {
              setIsCalling(true);
              setCallInitiator(true);
            }}>📞</button>
            </div>
          </div>
```

У нижній частині інтерфейсу розташоване поле вводу. У ньому розміщені кнопка вибору файлу з пристрою, текстове поле, а також кнопка «надіслати» (лістинг 3.7). За умови вибраного файлу або повідомлення для відповіді також з’являються відповідні модальні вікна, які дозволяють користувачу попередньо переглянути зміст того, що він намагається відправити (лістинг 3.8).

Лістинг 3.7 – Стилізація нижньої частини інтерфейсу чату

```
<div className="chat-input">
          <input type="file" ref={fileInputRef}
          style={{ display: 'none' }}
          onChange={handleFileSelect} />
```

```
          <button onClick={() => fileInputRef.current.click()}>+
          </button>
            <input
              type="text"
              placeholder="Enter the message..."
              value={messageInput}
              onChange={(e) => setMessageInput(e.target.value)}
            />
            <button
              onClick={handleSendMessage}>Send</button>
          </div>
```

Лістинг 3.8 – Стилізація вікна попереднього перегляду передачі файлів

```
{selectedFile && !(selectedFile.type.startsWith('image/')) && (
            <div className="file-preview">
              <span
         className="file-name">{selectedFile.name}</span>
              {' '}
              <a href={URL.createObjectURL(selectedFile)}
                 download={selectedFile.name}
                 className="file-download">⬇️</a>
        <button onClick={cancelFile}
                 className="file-cancel">×</button>
            </div>
          )}
          {previewUrl && selectedFile &&
selectedFile.type.startsWith('image/') && (
      <div className="file-preview">
              <img src={previewUrl}
                   alt={selectedFile.name}
                   className="image-preview"/>
         <button
           onClick={cancelFile}
           className="file-cancel">×</button>
      </div>
          )
         }
```

Поміж цими частинами розташоване вікно з повідомленнями. Воно зроблене як і у інших месенджерів: у невеличкій округлій фігурі знаходиться текст та/або медіазміст повідомлення. Якщо меседж вхідний, то поряд з ним розташовується аватар розмовника (при натисненні на нього як і у випадку з даними у верхній частині відкривається модальне вікно з даними користувача).

3.3 Опис програми: додатковий функціонал

Відображення повідомлень – звична річ, тому розберемо більш елегантну частину – розділювач дат. Він реалізований наступним чином: спочатку береться список повідомлень в чаті. Так як за допомогою WebSocket час і дата повідомлень записуються одразу після відправки, то з кожного з меседжів можна окремо взяти дату. Потім список сортується за датою. Далі йде порівняння: якщо у двох повідомлень різна дата, то між ними встановлюється розділювач, і у ньому записується дата того повідомлення, що було написано пізніше. Приклад такої реалізації зображено у лістингу 3.9, а відображення є на рисунку 3.2.

Лістинг 3.9 – Функція та відображення розділювача дат

```
{sortedMessages.map((msg, idx) => {
  const now = parseISO(msg.timestamp);
  const prev = idx > 0 ? parseISO(sortedMessages[idx -
1].timestamp) : null;
  const showSeparator = idx === 0 || !isSameDay(now, prev);
  const isOutgoing = msg.senderId === currentUserId;
     return (
        <React.Fragment key={msg.id}>
          {showSeparator && (
            <div className="date-separator">
               {format(now, 'MMMM d, yyyy')}
            </div>
     )})}
```

Рисунок 3.2 – Вигляд розділювача у застосунку

Також у застосунку є декілька функцій, пов’язаних з повідомленнями – їх прикріплення, видалення, а також відповідь на них. Оскільки ці дії з точки зору розробки є однотипними, детально розповідь йтиме лише про одну дію – прикріплення. Алгоритм відносно простий: для початку необхідно викликати функцію. У застосунку це реалізовано за допомогою пункту в контекстному меню (лістинг 3.10). Після натискання на пункт «Pin» надсилається запит на сервер з ідентифікатором вибраного повідомлення. Якщо меседж раніше був незакріплений – він закріплюється шляхом передачі у відповідний список (якщо меседж перший такий в чаті, то з’являється кнопка зі списком закріплених повідомлень), а стан оновлюється в реальному часі. Повторний такий запит робить обернену дію, тобто відкріпляє повідомлення.

Лістинг 3.10 – Стилізація контекстного меню з функцією прикріплення

```
{contextMenu.visible && (
           <ul className="message-context-menu"
           style={{ top: contextMenu.y, left: contextMenu.x }}>
           {!pinned.includes(contextMenu.messageId)
            ? <li onClick={() => {
                onPin(contextMenu.messageId);
                setContextMenu(c => ({ ...c, visible: false }));
            }}>Pin</li> :
              <li onClick={() => {
                onUnpin(contextMenu.messageId);
                setContextMenu(c => ({ ...c, visible: false }));
            }}>Unpin</li>}
           <li onClick={() => setContextMenu(c => ({ ...c,
visible: false }))}> Cancel </li>
           </ul>
         )}
```

У застосунку підтримується форматування повідомлень за допомогою Markdown. Так як це реалізовано за допомогою бібліотеки, було достатньо обернути зміст меседжу у елемент <ReactMarkdown>, що дозволяє зчитувати стилі, що були закладені у внутрішні поля. Тут і знадобляється санітаризація HTML-контенту, щоб зменшити кількість небезпечних комбінацій стилів.

У функціонал застосунку закладена можливість дзвінків іншому

користувачу. Для встановлення і підтримки зв’язку використовуються WebRTC та P2P як одна з найдоступніших технології організації викликів.

Щоб технологія справно працювала, для неї потрібно прописати сигнали, що надсилають користувачі:

- offer: надсилається у момент, коли користувач ініціює спробу виклика іншому. Поки не надійшов успішний сигнал Answer, з’являється вікно попереднього перегляду виклику, де можна перевірити, які потоки надсилаються;

- answer: надсилається у момент прийняття виклика приймаючою стороною. Фактично цей сигнал і є початком дзвінка;

- ICE (Interactive Connectivity Establishment) candidate: є сигналом визначення маршруту між двома peer. Скоріше є технічним засобом;

- end: надсилається у момент завершення чи неприйняття виклику.

Для того, щоб користувачі могли спілкуватись телефоном, обидва повинні бути в мережі. Тому було додано функціонал, який відстежує активність користувача. Це робиться за наступним алгоритмом: при реєстрації на сервері ініціалізується параметр, що відповідає за час останнього перебування в мережі. Він оновлюється кожний раз, коли користувач увійшов у мережу та вийшов з неї, і, дякуючи WebSocket, це оновлення відбувається в реальному часі. Простіше кажучи, людина зможе вчасно дізнатись, можна дзвонити користувачу чи ні. У лістингу 3.11 відбувається встановлення статусу користувачів після входу.

Лістинг 3.11 – Функція оновлення поточного статусу користувача

```
if (type === 'init') {
// Визначаємо користувача
   const uid = parseInt(payload.userId, 10);
   thisUserId = uid;
   clients.set(uid, ws);
   broadcast({
     type: 'userStatus',
     payload: { userId: uid, online: true, lastSeen:
uid.lastSeen}
   });
```

```
    const users = loadUsers();
    const statusList = users.map(u => ({
      userId:   u.id,
      online:   clients.has(u.id),
      lastSeen: u.lastSeen || null   }));
    ws.send(JSON.stringify({ type:'bulkStatus', payload:
statusList }));
    return;
}
```

З боку клієнта залишається лише зчитати статус (лістинг 3.12) та відобразити його (рисунок 3.3). У лістингу не просто зчитується дата останньої активності, а ще і на базі цих даних розраховується, скільки часу тому користувач був в мережі. Наочний приклад користі сторонніх бібліотек (в цьому випадку це formatDistanceToNow з date-fns).

Лістинг 3.12 – Функція відображення статусу користувача в мережі

```
const statusText = useMemo(() => {
  if (!selectedChat) return '';
  const partnerId = String(
    selectedChat.user1 === currentUserId
      ? selectedChat.user2
      : selectedChat.user1
  );
  const st = userStatuses[partnerId]||{ online: false, lastSeen:
null };

  if (st.online) {
    return '• Online';
  }
  if (st.lastSeen) {
    const last = new Date(st.lastSeen);
    if (!isNaN(last)) {
  return `Last seen ${formatDistanceToNow(last, { addSuffix:
true })}`;
    }
  }
  return 'Offline';
  }, [selectedChat, currentUserId, userStatuses]);
```

Рисунок 3.3 – Відображення статусу в застосунку (текст «Last seen»)

Тепер можна і спробувати подзвонити. Після того, як перший користувач здійснює спробу виклика, а інший ще не встиг його прийняти, у обох з’являється вікно попереднього перегляду (лістинг 3.13). Це означає, що за бажання користувач може перед дзвінком вимкнути передачу аудіо- чи відеопотоку. Ці QoL-функції допомагають людині виглядати з кращого боку.

Лістинг 3.13 – HTML-вигляд вікна попереднього перегляду дзвінка

```
{/* Інформація, що стосується дзвінка на прикладі вихідного */}
{outgoingCall && !inCall && !incomingCall && (
       <div className="outgoing-call">
         <p>📞 Calling {getDisplayName()}</p>
         <video ref={localRef} autoPlay playsInline muted
className="preview-local-video" /> {/* Тут відеопотік */}
       </div>)}
{/* Панель керування дзвінком */}
{(inCall || outgoingCall || incomingCall) && (
       <div className="controls">
         <button onClick={toggleMic}>Toggle Mic</button>
         <button onClick={toggleCamera}>Toggle Camera</button>
         <button onClick={toggleScreenShare}>Share
Screen</button>
         <button onClick={setFullscreen}>Fullscreen</button>
         <button onClick={enablePiP}>Enable PiP</button>
         <button onClick={handleEndCall}>End Call</button>
        </div>
        )}
```

Після прийняття другим користувачем дзвінок у вікні відбувається передача потоків між абонентами і вони можуть бачити і чути одне одного (за умови, якщо вони забажали передавати відповідні дані). Панель керування нікуди не зникає і користувач все ще може за бажанням перемкнути мікрофон чи камеру. Також стає доступним функціонал показу власного екрану.

4 ТЕСТУВАННЯ ПРОГРАМИ, СИСТЕМНІ ВИМОГИ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Тестування програми

Тепер важливо переконатися в тому, що розроблений застосунок буде доступним та стабільним для того, щоб його використовували звичайні люди. Для цього потрібно перевірити наступні речі:

- системні вимоги: не всі користувачі можуть собі дозволити новітню систему, особливо якщо стоїть мета використовувати месенджер. Тому важливо розробити застосунок максимально доступним з цієї точки зору;
- наявність і характер помилок: необхідно знайти якомога більше проблем, і виправити хоча б найбільш критичні з них. Такі помилки порушують роботу всіх компонентів, зокрема безпекових, тому головна мета – не допустити критичний експлойт до публічної версії застосунку.

4.1.1 Системні вимоги

Даний застосунок тестувався на такій системі: процесор AMD Ryzen 5 5500U з базовою частотою ядра 2.10 GHz; графічний процесор – інтегрований AMD Radeon Graphics; 8 ГБ оперативної пам’яті; операційна система: 64-бітна Windows 11 версії 23H2.

Вузького місця з боку апаратного забезпечення помічено не було, тому мінімальні вимоги будуть приблизно ті самі, що і для запуску браузеру. Наприклад, для цільового для розробки Google Chrome в залежності від операційної системи це:

- Windows: Windows 10 та новіше, або Windows Server 2016 та новіше у випадку з серверами;
- MacOS: macOS Big Sur 11 та новіше;

- Linux: 64-бітні Ubuntu 18.04+, Debian 10+ або Fedora 39+.

Для систем на Windows або Linux також мінімальною вимогою для запуску браузеру є процесор з підтримкою важливих інструкцій SSE3. Прикладами таких процесорів є Intel Pentium або AMD Athlon x64 2005 року випуску або пізніше. Вимогою до оперативної пам’яті у системі є наявність хоча б 4 ГБ, оскільки більшість з цієї пам’яті займе операційна система та ключові для її використання процеси.

Якщо використовувати інші браузери, як от Opera, Microsoft Edge чи Firefox, потрібно дивитись на їх системні вимоги. Але так як більшість браузерів написані на базі рушія Chromium, який фактично є публічною для розробки версією Chrome, вимоги не будуть значно відрізнятись.

4.1.2 Результати тестування та знайдені помилки

При тестуванні роботи в усіх браузерах було знайдено деякі технічні помилки, які не були передбачені при розробці месенджеру: при використанні кнопки “Delete Avatar” відсутність аватару не зберігається у профілі користувача; при спробі дзвінка у другого абонента не з’являється вікно, якщо він цілеспрямовано не чекає дзвінка в чаті, а також деякі інші помилки, що перешкоджають комфортній роботі у застосунку. Всі вже є в черзі на виправлення.

Було помічено і деякі візуальні недоліки: наприклад, лічильник непрочитаних повідомлень в елементі чату може не зникнути миттєво, а потребуватиме перезавантаження, а також не завжди записаний одразу після реєстрації username відображається при редагуванні профілю.

Також застосунок був протестований у різних браузерах на продуктивність та наявність помилок.

Google Chrome є цільовим браузером для розробки, тому основний фокус при виправлені помилок був саме на ньому. Для однієї вкладки застосунку витрати оперативної пам’яті склали від 80 МБ до 120 МБ в

залежності від сценарію використання, а також кількості повідомлень та файлів. Це є приблизними витратами будь-якої іншої стандартної вкладки, наприклад, перегляду документу.

Інші браузери на базі Chromium, такі як Opera та Microsoft Edge, мають приблизно той самий результат. Витрати пам’яті за вкладку склали ті ж 80-120 МБ. Оскільки ці браузери мають однакову «браузерну ДНК» з Chrome, то отримання будь-якого іншого результату було б аномальним явищем.

Firefox від Mozilla є одним з небагатьох браузерів, що не використовують для свого функціонування Chromium, тому у нього місцями буде трохи інший принцип роботи. Наприклад, через власну реалізацію смуги прокручування, переповнення частини вікна елементами чату не створює scrollbar, тобто дістатися до більш старих чатів у списку буде складніше. Також у Firefox є проблеми з відображенням шрифтів через їх власну реалізацію у браузері, що можна вважати за візуальну проблему. Витрати оперативної пам’яті склали від 100 до 200 МБ, тобто більші, ніж у браузерів на базі Chromium.

Наразі мобільні версії браузерів жодним чином не підтримуються. Застосунок в будь-якому з них зможе запуститись, у ньому можна буде навіть пройти реєстрацію та авторизацію, але подальші кроки можуть бути або неможливими, або складними для виконання. Тому інструкція користувача буде складена лише для користувачів будь-якою десктопною версією браузера.

4.2 Інструкція користувача

Крім знання технічної складової, такої як вимоги та наявні помилки, потрібно зробити так, щоб навіть найменш обізнана в технологіях людина могла розібратись у роботі застосунку і активно використовувати його в побуті. Це і є основною причиною складання інструкції користувача.

4.2.1 Початок використання: логін, реєстрація, додавання чату

Після запуску застосунок матиме вигляд як на рисунку 4.1. Натискання на кнопку «Register» змінить поточне вікно на те, що зображено на рисунку 4.2.

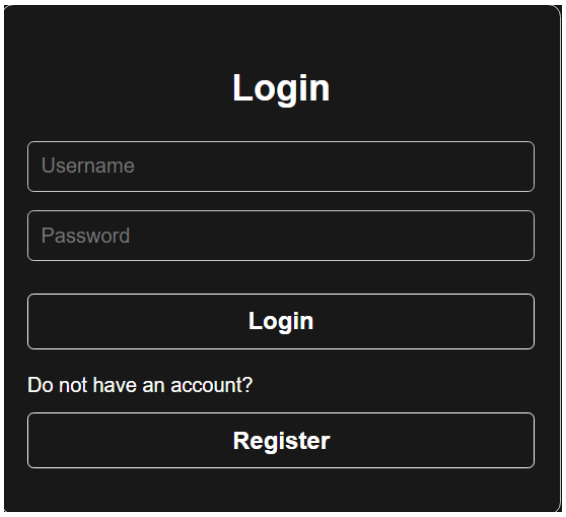

Рисунок 4.1 – Вікно логіну

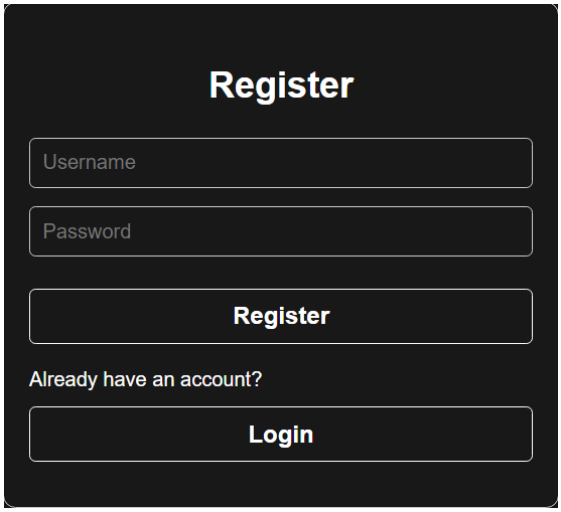

Рисунок 4.2 – Вікно реєстрації

Вводимо бажані для реєстрації дані, і якщо після натискання кнопки «Register» у цьому вікні повертається повідомлення «Registration successful», то реєстрація відбулася успішно, і можна повернутися у вікно логіну. Тоді можна у вікні логіну повторно ввести дані, за якими користувач

зареєстрований. У випадку успіху користувача відправляє у основний інтерфейс застосунку (рисунок 4.3).

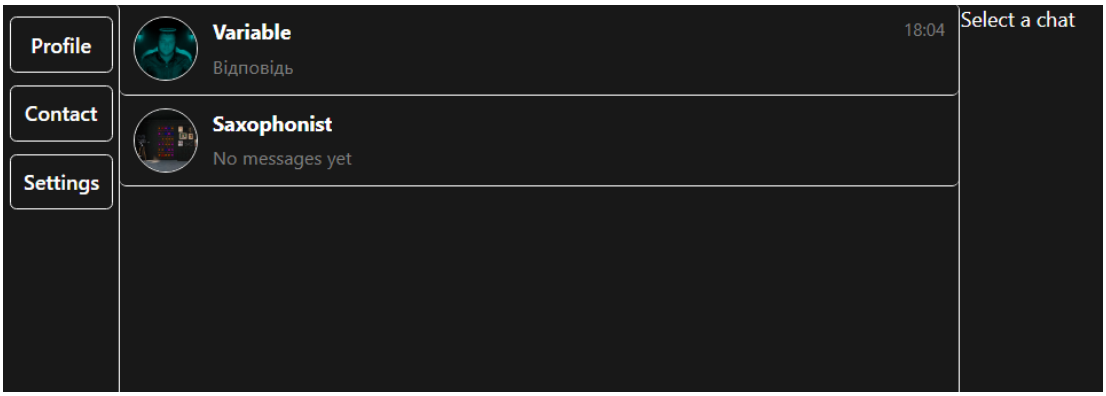

Рисунок 4.3 – Основний інтерфейс застосунку

Почнемо з персоналізації. Для цього потрібно натиснути на кнопку «Profile». Після натискання відкриється модальне вікно (рисунок 4.4) з можливістю редагувати необхідну інформацію.

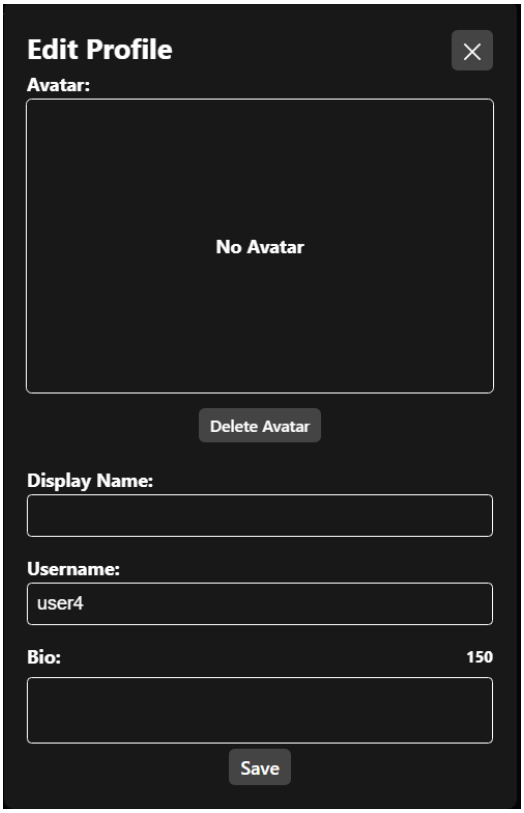

Рисунок 4.4 – Модальне вікно зміни профілю

У вікні можна змінювати ім’я для відображення (display name), ім’я користувача (username), інформацію про себе (bio), а також аватар. Якщо змінювати зміст текстового поля, то достатньо ввести потрібні дані. Для зміни аватару потрібно натиснути на вікно, де має відображатись аватар і обрати файл з пристрою. Після внесення змін потрібно натиснути кнопку збереження («Save»).

При натисненні кнопки налаштувань відкриється відповідне модальне вікно (рисунок 4.5). У ньому можна обрати тему застосунку – світлу, темну або системну. Системна тема повторює тему пристрою, на якому застосунок запущений.

Рисунок 4.5 – Модальне вікно налаштувань

Аби додати чат, потрібно натиснути на кнопку «Contact» – тоді відкриється модальне вікно додавання чату (рисунок 4.6). У вікні username потрібно ввести ім’я користувача, з яким необхідно встановити контакт. Після натиснути кнопку «Add».

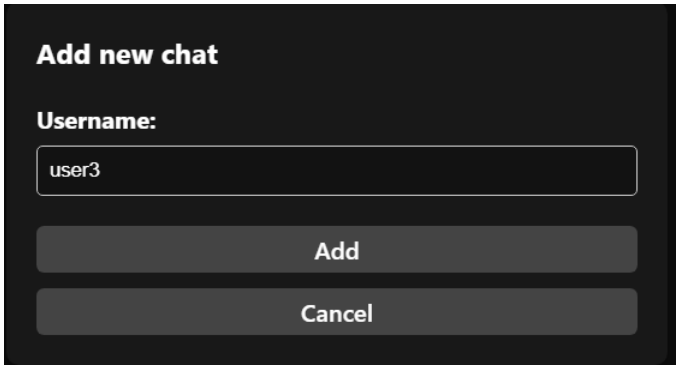

Рисунок 4.6 – Модальне вікно додавання чату

Якщо додавання успішне, з’явиться елемент списку чату (рисунок 4.7).

Рисунок 4.7 – Елемент списку чату

Цей елемент списку чату містить вкрай велику кількість інформації в собі. Але розберемо його по порядку (рисунок 4.8).

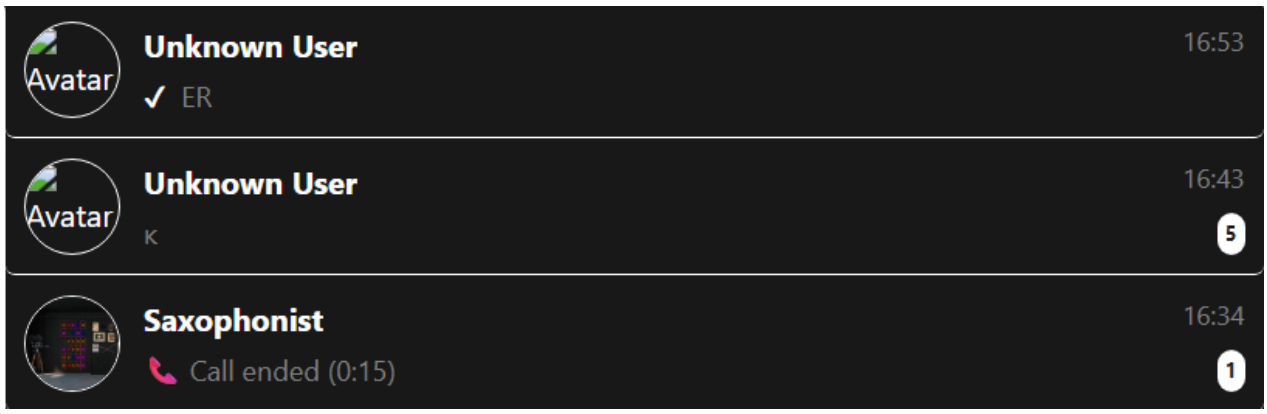

Рисунок 4.8 – Приклади інформації в елементах вікна чату

Кожен елемент обов’язково містить в собі аватар користувача (коло зліва), його ім’я для відображення (зверху зліва), час відправки/отримання (зверху справа), текст останнього повідомлення (знизу зліва).

Деякі елементи з’являються у вікні в залежності від контексту повідомлення. Справа знизу в елементі, за умови того, що в чаті є хоча б одне вхідне та одночасно з цим непрочитане повідомлення, з’являється лічильник непрочитаних в чаті повідомлень.

Якщо повідомлення було вихідним, то біля нього з’являється статус (чи було воно відправлено та прочитано співпозмовником). Статус відправленого повідомлення позначається як «✓», прочитаного – як «✓✓»

Також повідомленням є результат дзвінка, і з нього можна дізнатися

про спробу дзвінка, а також, якщо дзвінок був успішний, його тривалість.

Якщо натиснути на елемент, то відкриється вікно чату (рисунок 4.9). У ньому можна написати повідомлення у вікні внизу, а також всередині вікна є історія повідомлень. Про це ми ще поговоримо детальніше.

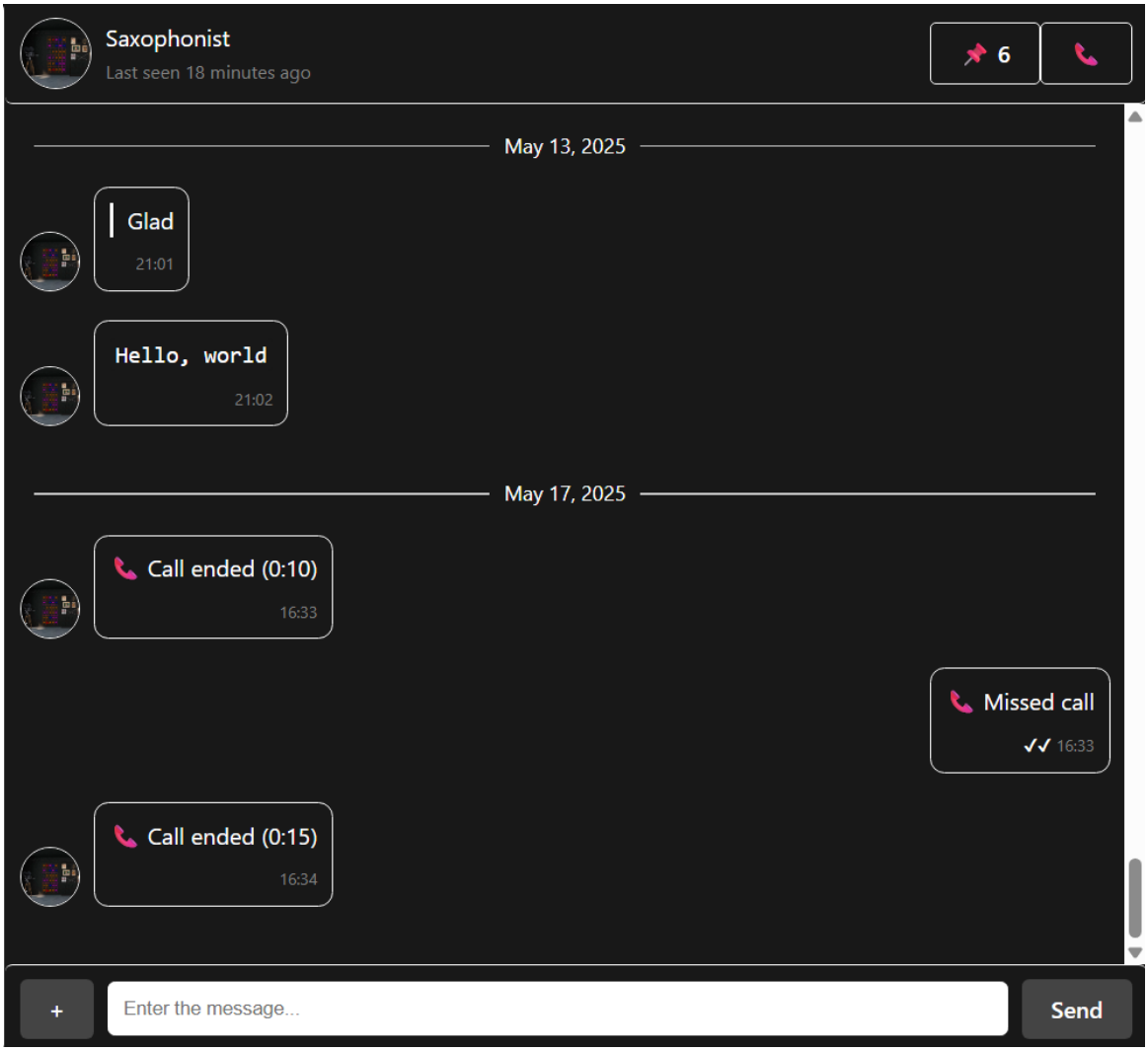

Рисунок 4.9 – Вікно чату

Почнемо з верхньої частини. В ній є інформація про співрозмовника, чат з ким був відкритий. Коло зліва відповідає за його аватар, зліва зверху пишеться його ім’я для відображення. Зліва знизу інформація про поточний статус користувача: або він у мережі «Online», або не в мережі «Last seen…». Якщо натиснути на верхню частину з лівого боку, відкриється модальне вікно з інформацією про співрозмовника (рисунок 4.10). В ньому

відображається детальна інформація про те, як ваш співрозмовник оформив свій профіль: його аватар, ім’я для відображення, нікнейм та статус.

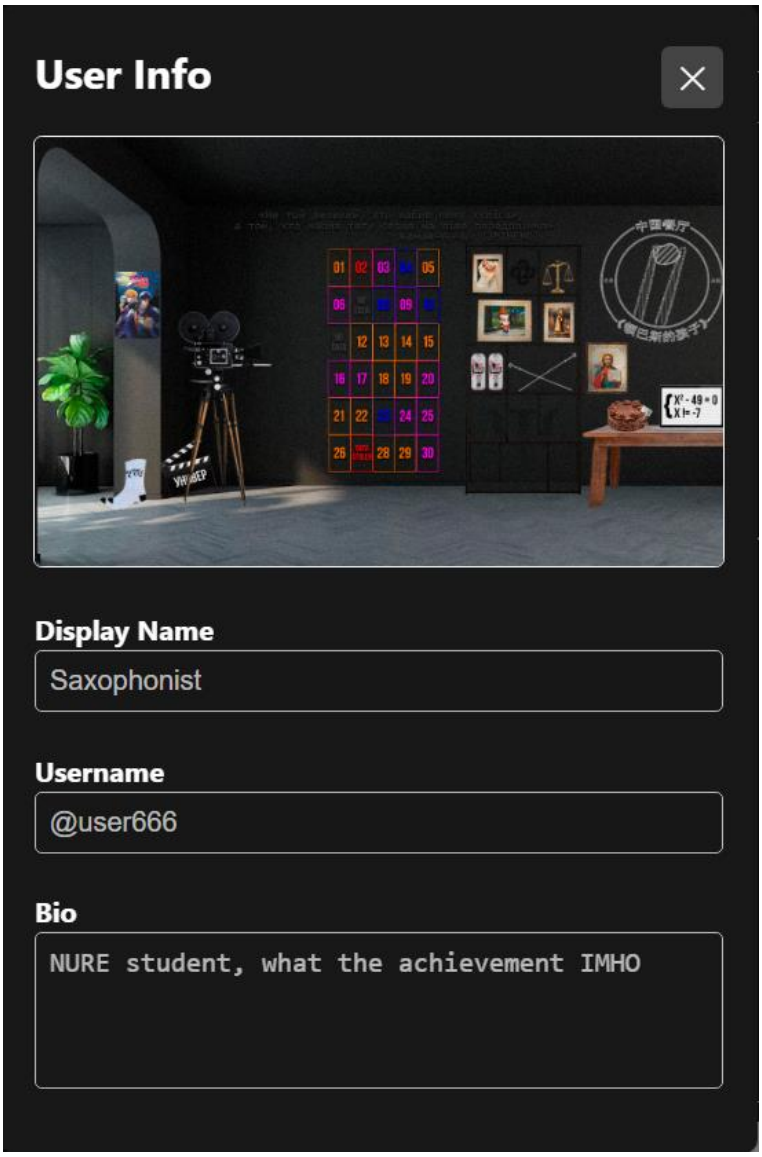

Рисунок 4.10 – Модальне вікно з інформацією про користувача

Права сторона верхньої частини відповідає за деякі функції спілкування: перша кнопка із зображенням канцелярської шпильки відповідає за відкриття списку закріплених в чаті повідомлень, друга, із зображенням слухавки, відповідає за те, щоб зателефонувати співрозмовнику. Про їх функціонал поговоримо трохи пізніше.

Для відправлення повідомлення необхідно подивитись в нижню частину вікна. Там є поле вводу, в яке можна ввести зміст повідомлення та

надіслати співбесіднику, натиснувши кнопку «Send». Проте звичайними повідомленнями вже абсолютно нікого не здивуєш. Тому повідомлення набрали декілька корисних функцій.

4.2.2 Просунуте використання

Перша з набутих функцій – надсилання файлу будь-якого формату. Для цього потрібно натиснути кнопку «+» зліва від поля вводу (рисунок 4.11) і обрати необхідний файл у вікні провіднику. Файл може бути будь-якого розширення, і за необхідності можна передати навіть пошкоджений.

Після вибору файлу над полем вводу з’явиться віконце попереднього перегляду. У ньому можна переглянути зміст файлу для потенційної відправки. Ще у цьому віконці файл можна завантажити, щоб повторно перевірити його перед відправкою, а також скасувати відправку, якщо це необхідно. Особлива увага приділяється поширеним форматам, таким як світлини, аудіозаписи та відеофайли. Їх легше передивлятись, оскільки зміст одразу відображається.

Рисунок 4.11 – Вікно попереднього перегляду файлу

Також весь текст підтримує основний функціонал Markdown, а повідомлення – підтримують розмітку і відображаються відповідним чином. Список стилів, що гарантовано підтримується в застосунку наступний:

- заголовки («#») будь-якого рівня;
- жирний і курсивний текст в будь-яких варіаціях («*» та «_»);
- фрагменти коду («`»);

- цитування («>») будь-якого рівня;
- посилання (як на вебсторінку, так і на зображення).

Вигляд стилізованих за допомогою Markdown повідомлень зображено на рисунку 4.12.

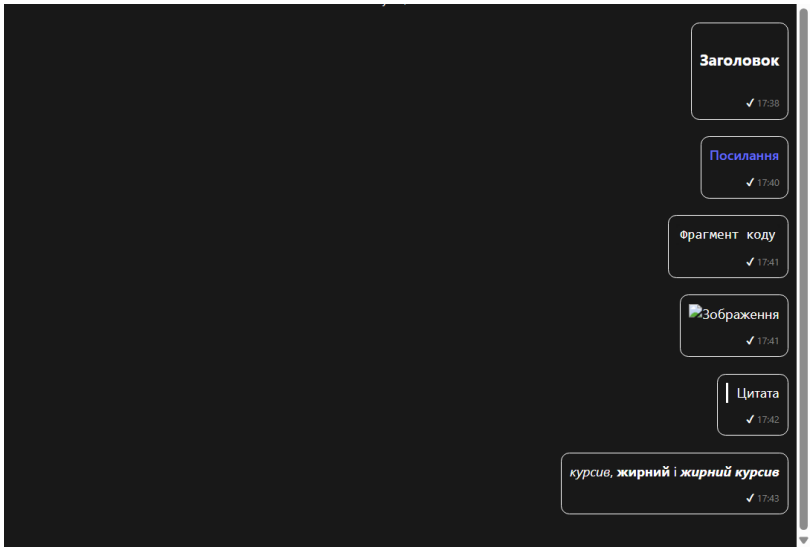

Рисунок 4.12 – Вікно чату з Markdown-стилізованими повідомленнями

До повідомлень було додано взаємодію з ними: їх можна видалити, прикріпити, а також на них можна відповісти. Аби дістатися цих функцій, необхідно відкрити контекстне меню повідомлення (рисунок 4.13), натиснувши правою клавішею миші на нього.

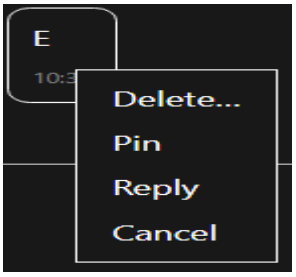

Рисунок 4.13 – Контекстне меню повідомлення

При спробі відповіді на повідомлення з’явиться віконце попереднього перегляду (рисунок 4.14). У ньому є інформація про надсилача повідомлення і його зміст, будь-то текстовий чи у вигляді файлового вкладення.

Рисунок 4.14 – Вікно попереднього перегляду відповіді на повідомлення

За необхідності можна скасувати відповідь на повідомлення, як і у випадку з файлом.

Після надсилання відповіді у повідомленні вказується, що воно носить такий характер і матиме вигляд, як на рисунку 4.15. Стиль відповіді є гіперпосиланням, тобто при натисненні на нього ви перейдете до оригінального повідомлення в чаті.

Рисунок 4.15 – Вікно чату з повідомленням-відповіддю

Також повідомлення можна прикріплювати. Для цього у контекстному меню потрібно обрати команду «Pin». Після цього лічильник закріплених повідомлень збільшиться, а в списку (рисунок 4.16) з’явиться повідомлення, що було щойно закріплено. Стиль повідомлення в списку закріплених також є гіперпосиланням і при натисканні користувач перейде до оригінального повідомлення.

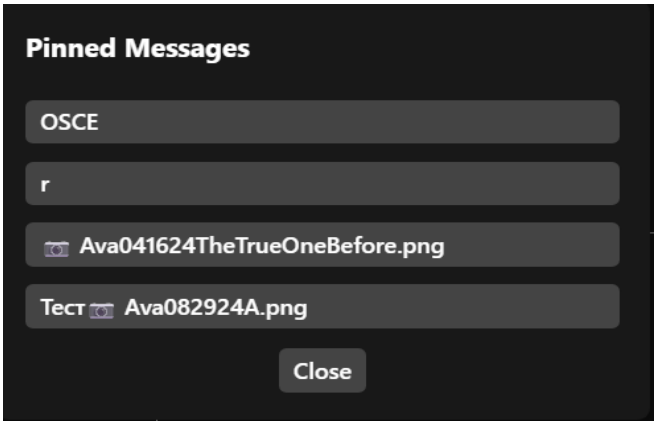

Рисунок 4.16 – Модальне вікно зі списком закріплених в чаті повідомлень

Крім цього, непотрібні повідомлення можна видалити. Для цього у контекстному меню потрібно обрати команду «Delete». При натисненні з’являється діалогове вікно (рисунок 4.17) в якому треба обрати, для кого видалити повідомлення – для себе чи для всіх в чаті (видалення для всіх доступне лише для власних повідомлень). Також можна скасувати видалення кнопкою «Cancel».

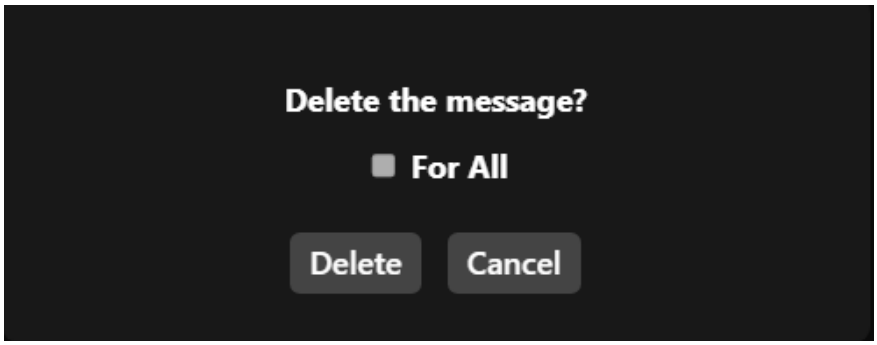

Рисунок 4.17 – Діалогове вікно для підтвердження видалення

Від повідомленнь перейдемо до ключової функції застосунку – дзвінків. Для того, щоб поздвонити, потрібно натиснути кнопку зі зображенням слухавки у верхній частині екрану. Через деякий час з’явиться вікно з повідомленням, що ви дзвоните абоненту (рисунок 4.18).

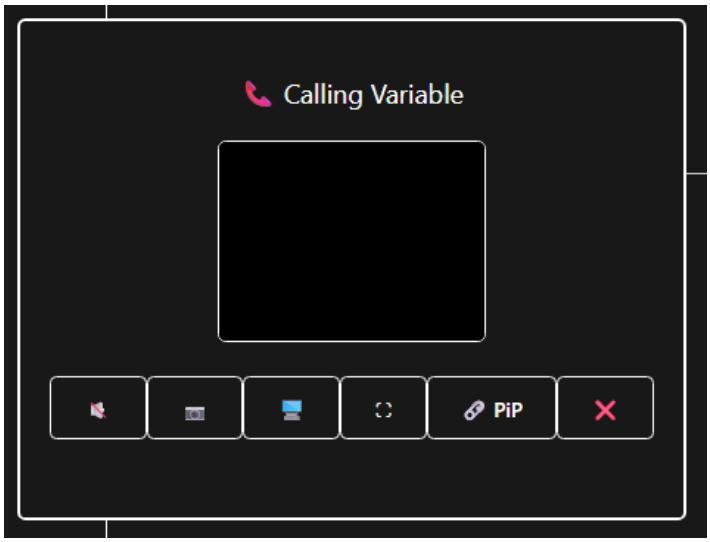

Рисунок 4.18 – Вікно дзвінка

В цьому вікні цікавить те, що воно також використовує попередній перегляд. За необхідності перед дзвінком можна відімкнути передачу звуку чи відео в дзвінок, або навіть скасувати його.

Важливо: щоб звук та відео працювали, браузер може попросити користувача надати необхідні дозволи для використання цих потоків (рисунок 4.19). Це було зроблено виключно з безпекових причин.

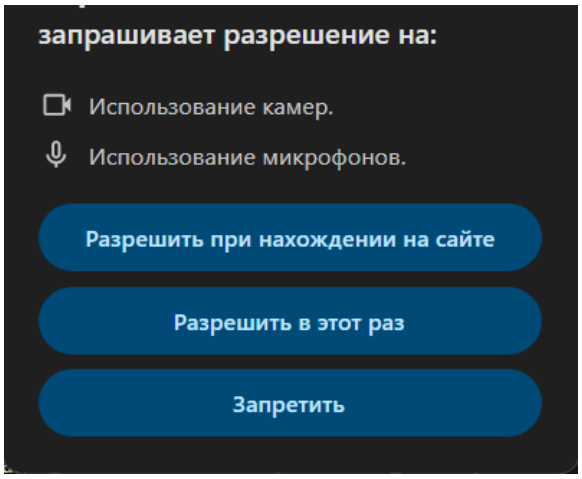

Рисунок 4.19 – Вікно запиту відповідних дозволів в Google Chrome

У другого абонента схоже вікно (рисунок 4.20), де він також може підготуватись перед дзвінком, а також за потреби прийняти чи скасувати його.

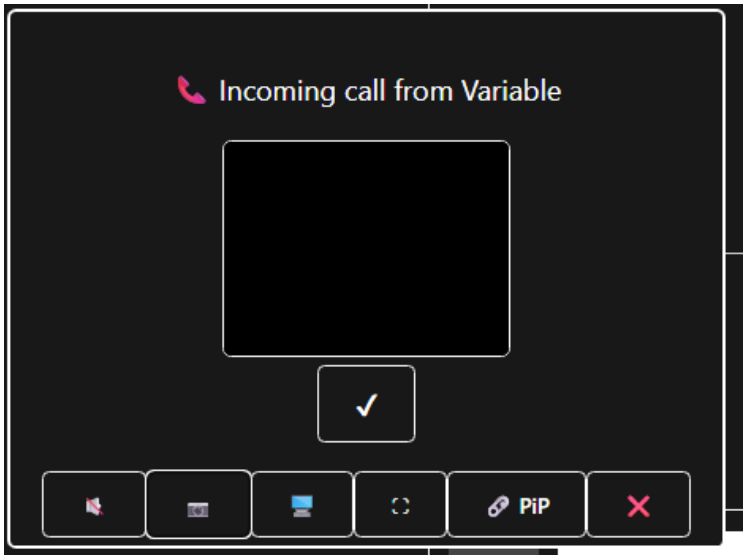

Рисунок 4.20 – Вікно дзвінка у другого абонента

Після прийняття дзвінка у обох користувачів вікно зміниться на те, що представлено на рисунку 4.21. Воно і є користувацьким інтерфейсом дзвінка.

Рисунок 4.21 – Вікно активного дзвінка

Розберемо функціонал цього вікна по порядку:

- у верхньому лівому куті знаходиться таймер, який заміряє, скільки часу триває поточний дзвінок;
- у верхньому правому куті знаходиться ваш відеопотік
- весь екран вікна займає відеопотік вашого співрозмовника;
- перша кнопка в нижній панелі – перемикання передачі звукового потоку. Перше натискання його вимикає, друге – вмикає;
- друга кнопка в нижній панелі – перемикання передачі відеопотоку. Як і у випадку зі звуком, перше натискання його вимикає, друге – вмикає;
- третя кнопка – передача співрозмовнику показу власного екрану. Підтримується передача як одного конкретного вікна чи вкладки браузера, так і всього екрану. Виглядає це як на рисунку 4.22;
- четверта кнопка – розширення вікна дзвінка до повноекранного. Особливо допоможе у випадках, коли потрібно добре бачити зміст показу екрану. Друге натискання повертає розміри вікна до звичних;

- п’ята кнопка дозволяє взяти показ вашого відео в режим «картинка в картинці»;

- шоста кнопка вкрай проста – вона завершує або скасовує виклик незалежно від того, чи він розпочався, чи ні.

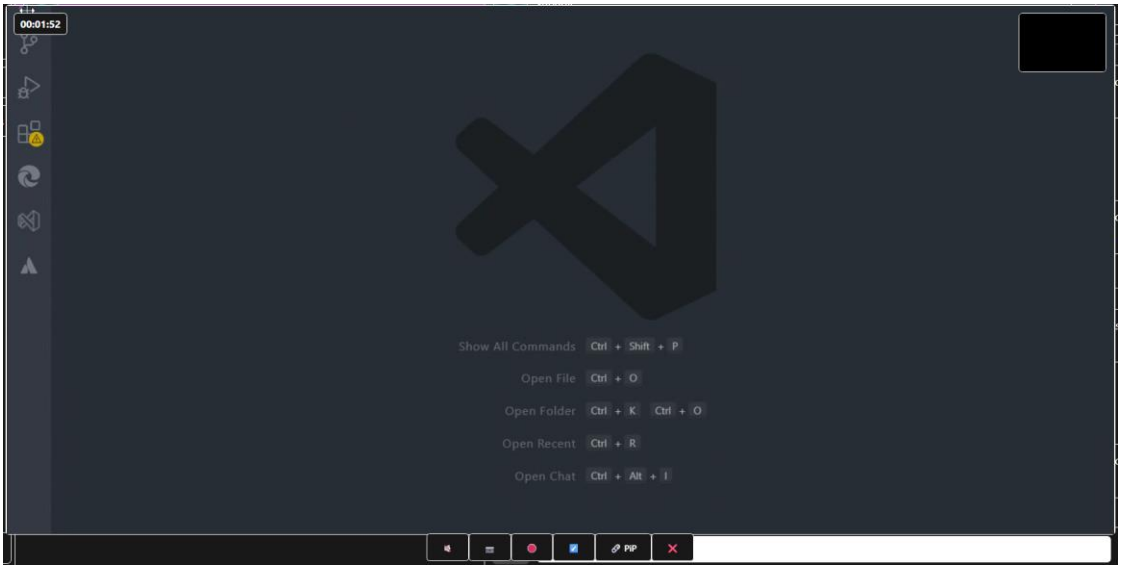

Рисунок 4.22 – Вікно активного дзвінка у повному екрані з показом екрана

Також було передбачено можливість планування робочого простору застосунку: можна виділити вікну чату стільки місця, скільки користувачу необхідно. Можна як збільшити його обсяг, так і зменшити аж до мобільного форм-фактору за допомогою перетягування розділювача списку і вікна чату за допомогою натиснутої лівої клавіші миші в ту чи іншу сторону. Мінімальний та максимальний розмір зображено на рисунках 4.23 та 4.24.

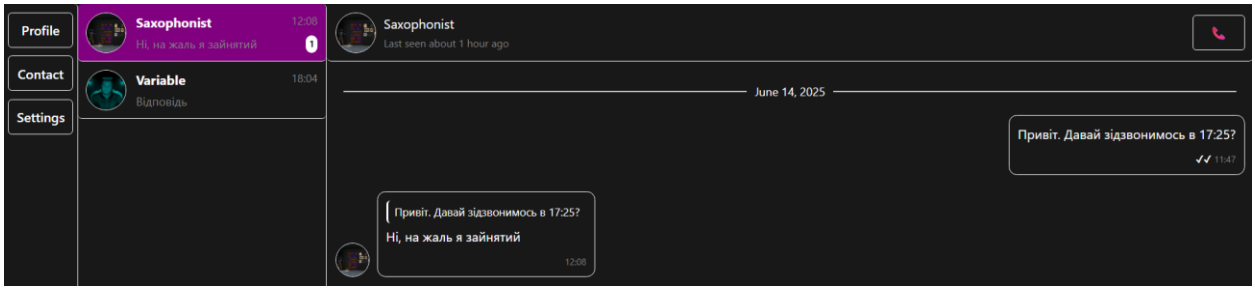

Рисунок 4.23 – Вікно чату, розтягнуте до максимуму

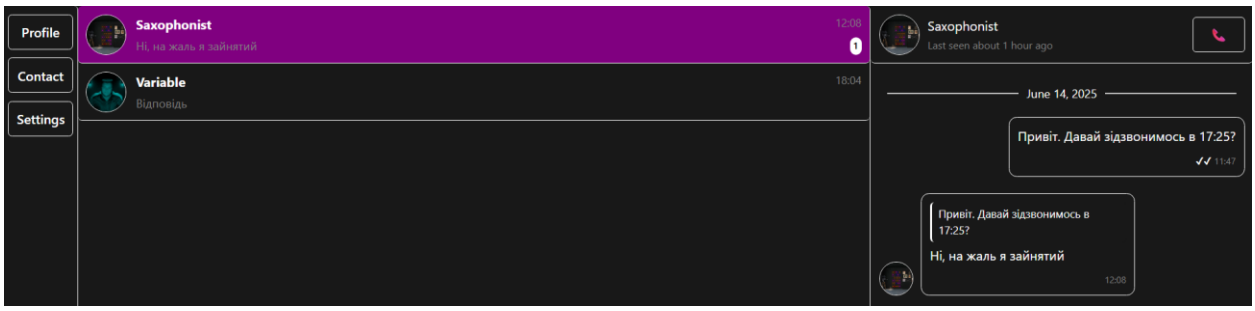

Рисунок 4.24 – Вікно чату, розтягнуте до мінімуму

Якщо більше не потрібно використовувати месенджер в поточній сесії, то в нижньому лівому куті є кнопка «Logout» (рисунок 4.25).

Рисунок 4.25 – Частина вікна з кнопкою «Logout»

Усі дані, що необхідні для продовження спілкування наступного разу вже записані за допомогою WebSocket та серверних запитів. Крім того, серверні дані інших користувачів оновлюються завжди, якщо з їх даними зробили релевантну дію (наприклад, написали повідомлення або прикріпили його в її чаті).

ВИСНОВКИ

В даній кваліфікаційній роботі було успішно розроблено клієнт-серверний вебзастосунок для спілкування між користувачами.

Для виконання було використано наступний стек: для клієнту використовувалась мова розмітки HTML, мова стилізації CSS, мова програмування JavaScript та фреймворк до неї React. Сервер був написаний за допомогою Node.js, а для вирішення залежностей і технічного зв’язку компонентів було використано менеджер пакетів NPM.

Завдання було виконано наступними кроками:

Спочатку було поставлено предметні задачі, які повинні вирішити проблеми месенджерів або хоча б запропонувати для них компроміс. Основними з них були розробити метод ідентифікації користувачів, необхідний для роботи функціонал, функції для комфортного використання месенджером, вирішити безпекову, функціональну та дизайнерську проблеми, а також робити це надійно і продуктивно.

Потім було обгрунтовано та обрано напрямок розробки та цільову платформу, якими відповідно стали веб-розробка та браузер. Згодом на базі інформації про напрям розробки було обрано стек програмування: описано кожний не лише обраний компонент, але і деякі аналогічні варіанти, а також обгрунтовано причину вибору саме цих компонентів.

Наступним кроком було остаточно сформовано архітектуру та структуру застосунку, якими способами всередині обраного стеку програмування потрібно вирішувати ті чи інші задачі на практиці, а також які саме надбудови використовувати разом з основними компонентами. Наприклад, для дзвінків в купі з WebSocket було прийнято рішення використовувати WebRTC.

Далі було закладено фундамент застосунку на практиці: остаточно визначено всі бібліотеки та залежності, що використовуватимуться в

застосунку, а також розроблено безліч основних функцій: процедури реєстрації та автентифікації, збереження облікових даних на сервері, завантаження профілів для виведення інформації, додавання контактів, відправку повідомлень, додавання налаштувань тощо.

Минулі кроки дозволили згодом додати функції, що спрощують використання застосунку звичайним користувачем. Серед таких є розділення повідомлень за датою їх написання, статуси користувача (в мережі та не в мережі) та повідомлень (доставлено та прочитано), взаємодія з повідомленнями, така як їх прикріплення або видалення, відправка файлів та їх підтримка у повідомленні, додання метапосилань для швидкого переходу до необхідних повідомлень, а також розроблена повноцінна система дзвінків.

Далі відбулося тестування. За його допомогою вдалося знайти та виправити багато помилок. Наприклад, була помилка, що у користувача спрацьовувало видалення вхідного повідомлення для всіх або що у дзвінку при відкритті власного відеопотоку у режимі «картинка в картинці» він не оновлювався. Було вказано ті помилки, які будуть в числі наступних для виправлення, а також визначено системні вимоги не лише для різних браузерів, але і для різних операційних систем.

Програма продемонструвала високу продуктивність для різних дій всередині неї, а також прийнятну для постійного використання безвідмовність і безпеку, і використовувала мінімум ключових для системи ресурсів. Також було виміряно витрати оперативної пам’яті застосунком в основних браузерах і в рамках проєкту було спростовано поширений міф, що Google Chrome використовує для своєї роботи багато основної пам'яті.

Наразі існує потенціал для майбутнього розвитку застосунку: поточна архітектура не готова приймати дуже великі потоки користувачів, у проєкті не вистачає багатьох важливих функцій, таких як налаштування приватності, пересилання у чати, групові чати, а вже існуючі фішки мають візуальні чи функціональні помилки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Hohpe G., Woolf B. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. 1st ed. Boston, MA : Pearson Education, 2003. 736 p.

2. Severance C. R. Introduction to Networking: How the Internet Works. Kindle ed. [s. l.] : Charles Severance, 2015. 151 p.

3. Mahapatra R. Analog and Digital Communication: Visualize and analyze the communication. 1st ed. New Delhi : BPB Publications, 2024. [без пагінації].

4. England R. Working in IT. Paperback ed. [s. l.] : Lulu Press, Inc., 2009. 116 p.

5. Threema – Private Messenger Comparison. URL: https://threema.com/en/products/private/messenger-comparison (дата звернення: 30.05.2025).

6. Orfali R., Harkey D., Edwards J. Client/Server Survival Guide. 3rd ed. New York, NY : John Wiley & Sons, 1999. 803 p.

7. Ciesla R. The Book of Chatbots: From ELIZA to ChatGPT. 1st ed. Cham : Springer, 2024. 173 p.

8. Johns A., Matamoros-Fernandez A., Baulch E. WhatsApp: From a one-to-one Messaging App to a Global Communication Platform. 1st ed. Cambridge : Polity, 2024. 220 p.

9. Telegram Tips. URL: https://t.me/TelegramTips (дата звернення: 31.05.2025).

10. Discord Developer Documentation. URL: https://discord.com/developers/docs/intro (дата звернення: 30.05.2025).

11. Butterfield S. Flickr Co-Founder Stewart Butterfield Turns to Workplace Communication Tools with Slack. URL: https://allthingsd.com/20130814/flickr-co-founder-stewart-butterfield-turns-to-workplace-communication-tools-with-slack/ (дата звернення: 31.05.2025).

12. Pollard B. HTTP/2 in Action. 1st ed. New York, NY : Manning, 2019. 416 p.

13. Pereyra I. Universal Principles of UX: 100 Timeless Strategies to Create Positive Interactions between People and Technology. Vol. 4. Beverly, MA : Rockport Publishers, 2023. 224 p.

14. Tharp A. L. File Organisation and Processing. 1st ed. New York, NY : Wiley, 1991. 416 p.

15. Smith J. Build Your Own Web Server From Scratch in Node.JS: Learn network programming, HTTP, and WebSocket by coding a Web Server. [s. l.] : James Smith, 2024. 131 p.

16. Preston W. C. Backup & Recovery: Inexpensive Backup Solutions for Open Systems. 1st ed. Sebastopol, CA : O'Reilly Media, 2007. 758 p.

17. Orosz G. Building Mobile Apps at Scale: 39 Engineering Challenges. [s. l.] : Primedia E-launch LLC, 2021. 236 p.

18. Adefioye T. Ultimate Flutter for Cross-Platform App Development. [s. l.] : Orange Education Pvt Ltd, 2024. 416 p.

19. Stolley K. Programming WebRTC: Build Real-Time Streaming Application for the Web. 1st ed. Raleigh, NC : Pragmatic Bookshelf, 2024. 268 p.

20. Wessels D. Web Caching: Reducing Network Traffic. 1st ed. Sebastopol, CA : O'Reilly Media, 2001. 318 p.

21. Dean J. Web Programming with HTML5, CSS, and JavaScript. Pap/Psc ed. Burlington : Jones & Bartlett Learning, 2018. 678 p.

22. Delamater M. Murach's Modern JavaScript: Beginner to Pro. 1st ed. Fresno, CA : Mike Murach and Associates Inc, 2024. 602 p.

23. Monteiro F. Learning Single-page Web Application Development. Birmingham : Packt Publishing, 2014. 216 p.

24. Bampakos A. Learning Angular: A practical guide to building web applications with modern Angular. 5th ed. Birmingham : Packt Publishing, 2025. 486 p.

25. Cuomo S. Vue.js 3 for Beginners. Birmingham : Packt Publishing, 2024. 302 p.

26. Wieruch R. The Road to React: Your journey to master plain yet pragmatic React.js. [s. l.] : Robin Wieruch, 2018. 286 p.

27. Freeman A. Mastering Node.js Web Development. Birmingham : Packt Publishing, 2024. 778 p.

28. Afonso D. State Management with React Query: Improve developer and user experience by mastering server state in React. 1st ed. [s. l.] : Packt Publishing, 2023. 228 p.

29. Martin R. C. Functional Design: Principles, Patterns and Practices. – 1st ed. Boston, MA : Addison-Wesley Professional, 2023. 384 p.

30. Personal Cloud Storage & File Sharing Platform – Google. URL: https://drive.google.com/file/d/1X35ytCx2YveQrGgAZazrvGTcqD1HEoCy/view?usp=sharing (дата звернення: 14.06.2025)