

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Вебмесенджер з файл-обмінником»

Виконав студент групи ПЗ-21-1
КЕРЕСТ Назар Іванович

Керівник роботи:
ЛОКТИКОВА Тамара Миколаївна

Рецензент:
ВАКАЛЮК Тетяна Анатоліївна

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного
забезпечення

Тетяна ВАКАЛЮК

«17» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КЕРЕСТ Назар Іванович**

Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**

Тема роботи: **«Вебмесенджер з файл-обмінником»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **об'єктом дослідження є процеси створення месенджера з файл-обмінником, а предметом дослідження є методи, засоби та алгоритми для побудови та проєктування вебмесенджера з покращеною автентифікацією та функцією обміну контентом.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Локтікова Т. М.	17.02.2025 р.	17.02.2025 р.
2	Локтікова Т. М.	17.02.2025 р.	17.02.2025 р.
3	Локтікова Т. М.	17.02.2025 р.	17.02.2025 р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебмесенджера “Connect” та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить текстову частину, викладену на 83 сторінках друкованого тексту, 56 ілюстрації, 15 таблиць. Список використаних джерел містить 25 найменувань і займає 2 сторінки. Також у роботі є 35 сторінок додатків. Загальний обсяг роботи складає 118 сторінок.

Виконано розробку вебмесенджера з файл-обмінником, який забезпечує листування з користувачами, можливість управління користувацькими групами.

У результаті огляду й аналізу можливих архітектурних рішень для системи було обрано архітектуру MVC.

Для розробки системи були використані сучасні вебтехнології – Next.js, NestJS, PostgreSQL, Twilio, сервіс Google.

Інтерфейс застосунку розроблено з урахуванням сучасних вимог до зручності та адаптивності, що забезпечує інтуїтивно зрозумілий процес обміну повідомленнями та взаємодії між користувачами.

Розроблений вебмесенджер пройшов необхідні види тестування.

КЛЮЧОВІ СЛОВА: ВЕБМЕСЕНДЖЕР, ФАЙЛ-ОБМІННИК, MVC, NEXT.JS, NESTJS, POSTGRESQL, TWILIO.

					ІПЗ.КР.Б-121-25-ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Керест Н. І.			Вебмесенджер з файл-обмінником. Пояснювальна записка	Лім.	Арк.
Перевір.		Локтікова Т. М.					3
Керівник		Локтікова Т. М.				Аркушів	118
Н. контр.		Тичина О.П.				ФІКТ Гр. ІПЗ-21-1	
Зав. каф.		Вакалюк Т. А.					

ABSTRACT

The bachelor's qualification thesis consists of the “Connect” web messenger and an explanatory report. The explanatory report contains a textual part presented on 83 pages of printed text, 56 illustrations, and 15 tables. The list of references includes 25 sources and occupies 2 pages. The thesis also includes 35 pages of appendices. The total volume of the work is 118 pages.

A web messenger with a file-sharing system has been developed, enabling user-to-user messaging and user group management.

As a result of the review and analysis of possible architectural solutions for the system, the MVC architecture was chosen.

Modern web technologies were used in the system development — Next.js, NestJS, PostgreSQL, Twilio, and Google services.

The application interface was developed in accordance with modern usability and adaptability standards, ensuring an intuitive messaging and interaction experience for users.

The developed web messenger has undergone the necessary types of testing.

KEYWORDS: WEB MESSENGER, FILE SHARING, MVC, NEXT.JS, NESTJS, POSTGRESQL, TWILIO.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБМЕСЕНДЖЕРА З ФАЙЛ-ОБМІННИКОМ .	9
1.1. Постановка задачі.....	9
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури вебмесенджера	18
1.4. Обґрунтування вибору інструментальних засобів	20
Висновки до першого розділу.....	24
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБМЕСЕНДЖЕРА	26
2.1. Визначення варіантів використання системи	26
2.2. Розробка бази даних.....	34
2.3. Розробка функціональних алгоритмів роботи програми.....	43
2.4. Реалізація вебмесенджера	45
Висновки до другого розділу	51
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБМЕСЕНДЖЕРОМ	53
3.1 Структура інтерфейсу вебмесенджера.....	53
3.2 Тестування системи	75
Висновки до третього розділу	79
ВИСНОВКИ.....	80
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	81
ДОДАТКИ.....	83

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

API – Application Programming Interface (прикладний програмний інтерфейс)

REST API – Representational State Transfer Application Programming Interface
(архітектурний стиль взаємодії клієнта з сервером через HTTP)

UI – User Interface (користувацький інтерфейс)

UX – User Experience (користувацький досвід)

CRUD – Create, Read, Update, Delete (базові операції з даними)

JWT – JSON Web Token (формат токена для аутентифікації)

QR-код – Quick Response Code (двовимірний штрихкод для швидкої
ідентифікації)

JSON – JavaScript Object Notation (формат обміну даними)

DOM – Document Object Model (об'єктна модель документа типу
HTML/XML)

CDN – Content Delivery Network (мережа доставки контенту)

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність теми пов'язана із тим, що в час, коли цифрові технології пронизують всі сфери життя, миттєвий обмін інформацією є критично важливим як для звичайних людей, так і для компаній. Зі збільшенням онлайн-комунікацій, віддаленої роботи та цифрової взаємодії, виникає гостра потреба в швидких, зручних, безпечних та сучасних інструментах для обміну повідомленнями. Вебмесенджери, як універсальні засоби зв'язку, користуються великою популярністю, але вимагають високої продуктивності, надійності та сумісності з сучасними вебтехнологіями. Створення власного вебмесенджера дає можливість детально вивчити розробку SPA-застосунків, роботу з WebSocket та організацію повноцінної взаємодії між клієнтом і сервером.

Метою роботи є проєктування та реалізація вебмесенджера “Connect” із файл-обмінником, який забезпечує обмін повідомленнями в реальному часі.

Для досягнення поставленої мети необхідним є виконання таких **завдань**.

1. Провести аналіз існуючих аналогів вебмесенджерів, зокрема дослідити їхні функціональні можливості, переваги та недоліки.
2. Обґрунтовано обрати методологію розробки, інструменти та технології для побудови й проєктування.
3. Реалізувати вебмесенджер на основі обраного технологічного стеку, спроектувати архітектуру та створити основні функціональні модулі.
4. Здійснити комплексне тестування з метою перевірки працездатності, стабільності та відповідності вимогам.

Об'єктом дослідження є процеси створення месенджера з файл-обмінником.

Предметом дослідження є методи, засоби та алгоритми для побудови та проєктування вебмесенджера з покращеною автентифікацією та функцією обміну контентом.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези: Керест Н.І., Локтікова Т.М., Петросян Р.В. Аналіз ключових вимог та особливостей реалізації веб-месенджера. Тези доповідей VII Всеукраїнської

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

науково-технічної конференції «Комп'ютерні технології: іновачії, проблеми, рішення», 02-03 грудня 2024 року. Житомир: «Житомирська політехніка», 2024. С. 67-68 [3].

Також за темою випускної кваліфікаційної роботи бакалавра подані та прийняті до опублікування тези: Керест Н.І., Локтікова Т.М., Лисогор Ю.І. РWA вебмесенджер з покращеною автентифікацією та функцією обміну контентом. Всеукраїнська науково-практична on-line конференція здобувачів вищої освіти і молодих учених, присвячена Дню науки, 12 травня 2025 року [2].

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБМЕСЕНДЖЕРА З ФАЙЛ- ОБМІННИКОМ

1.1. Постановка задачі

У сучасному цифровому середовищі спілкування між людьми відіграє ключову роль в особистому та професійному житті. Оскільки віддалена робота, навчання та онлайн-спільноти зростають, виникає потреба у зручних і безпечних способах спілкування. Існуюче програмне забезпечення для обміну миттєвими повідомленнями, таке як Telegram, WhatsApp, Viber або Facebook Messenger, має широкий спектр функціональних можливостей, але не завжди відповідає потребам певних груп користувачів або конкретним випадкам використання.

Замість того, аби зосереджуватися на потребах користувачів, багато сучасних продуктів перевантажені зайвими функціями, які роблять їх складними та незручними у використанні. Розробники повинні прагнути до простоти та гнучкості, пропонуючи рішення, які легко інтегруються з іншими сервісами та доступні на різних платформах, включаючи веб-версії та офлайн-режим. Відкриті API та підтримка різних форматів даних також є важливими для забезпечення сумісності та зручності використання. Лише зосереджуючись на реальних потребах користувачів та пропонуючи прості, гнучкі та адаптивні рішення, можна створити продукти, які дійсно будуть корисними та ефективними.

Оскільки існує нагальна потреба в простому, зручному, гнучкому та захищеному інструменті для онлайн-спілкування, пропонується розробити вебмесенджер “Connect”. Ця система має бути сучасною, швидкою та універсальною, але водночас інтуїтивно зрозумілою та доступною для широкої аудиторії. Основна мета – забезпечити користувачам можливість легко обмінюватися текстовими повідомленнями, фотографіями, відео, аудіо та файлами через максимально простий інтерфейс [4]. Ключовою перевагою є відсутність необхідності встановлювати будь-яке програмне забезпечення. “Connect” працюватиме безпосередньо у веб-браузері, що значно полегшить доступ і

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

залучення нових користувачів. Це особливо важливо для компаній, навчальних закладів, командних проєктів та тимчасових груп, де швидке та безперешкодне підключення до чата є критичним. Система повинна бути адаптована для роботи на різних пристроях, від комп'ютерів до мобільних телефонів, автоматично підлаштовуючись під розмір екрану та потреби користувача. Безпека є пріоритетом, тому шифрування повідомлень, надійна автентифікація та захист особистих даних повинні бути вбудовані в систему з самого початку [19]. Розроблювальний застосунок має на меті вирішити проблеми, пов'язані зі складними або обмеженими месенджерами, пропонуючи сучасне рішення, яке поєднує в собі простоту, доступність, інтеграцію та безпеку. Це крок до більш відкритої, зручної та орієнтованої на користувача цифрової комунікації.

Основна мета проєкту – розробка вебмесенджера “Connect”, який дозволить користувачам робити швидку, захищену та зручну комунікацію в реальному часі через веб-інтерфейс.

Цей вебмесенджер буде мати ряд важливих функцій, які зосереджені на зручності, безпеці та сучасному користувацькому досвіді. Користувачі зможуть легко зареєструватися і входити в систему за допомогою свого номера телефону – авторизація проходитиме через SMS-код, що забезпечить швидкий і надійний доступ без необхідності запам'ятовувати складні паролі. Основна функція передбачає обмін текстовими повідомленнями в реальному часі, що створює зручні умови для особистого або робочого спілкування. Крім того, користувачі зможуть передавати один одному різні файли – документи, зображення, відео та інші вкладення, які будуть зберігатися в історії переписки для подальшого доступу. Особливий акцент зроблено на дизайні та зручності інтерфейсу. Інтерфейс буде інтуїтивно зрозумілим навіть для новачків і включатиме привабливу анімовану вступну частину.

Буде створено вебмесенджер “Connect”, який запропонує більш ефективне рішення для комунікації у вузьких групах та організаціях, ніж існуючі масові месенджери. “Connect” вирізнятиметься простотою використання, високим рівнем конфіденційності, швидким доступом та широкими можливостями налаштування

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

під конкретні потреби. Це зробить його найбільш підходящим засобом для внутрішньої комунікації в компаніях, обміну повідомленнями в проєктних командах та спілкування у спеціалізованих спільнотах. Завдяки своїй гнучкості, “Connect” може бути інтегрований з іншими системами та адаптований до унікальних робочих процесів. Замість конкуренції з масовими платформами, “Connect” пропонує спеціалізоване рішення для тих, хто потребує безпечної, надійної та легко керованої комунікації, що враховує їхній контекст та потреби.

1.2. Аналіз аналогів програмного продукту

Перед початком розробки вебмесенджера необхідно провести всебічний аналіз відповідного ринку. Оцінка функціональних можливостей, переваг і недоліків існуючих продуктів дозволить мінімізувати ризики, створити конкурентоспроможне рішення, що відповідатиме актуальним потребам користувачів, а також визначити потенційні напрямки для інноваційного розвитку.

Одним з аналогів є Telegram Web [23] – це широко використовуваний застосунок для обміну повідомленнями, який зберігає дані в хмарі, забезпечуючи доступ з різних пристроїв. Він пропонує різноманітні функції, включаючи текстові чати, обмін файлами, групові розмови, канали для трансляцій та інтеграцію з ботами.

Веб-версія Telegram вирізняється інтуїтивно зрозумілим дизайном та швидкою роботою. Попри зручний інтерфейс та високу продуктивність, веб-версія Telegram може здатися занадто складною для початківців через надмірну кількість функцій.

Відзначимо переваги Telegram Web. Даний вебмесенджер пропонує низку можливостей для онлайн-комунікації, включаючи хмарне зберігання даних, що забезпечує доступ до історії повідомлень з різних пристроїв. Підтримка ботів дозволяє автоматизувати процеси та інтегрувати сервіси. Інтерфейс платформи розроблений з урахуванням зручності користувача.

На рис. 1.1, представлено головну сторінку Telegram Web.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

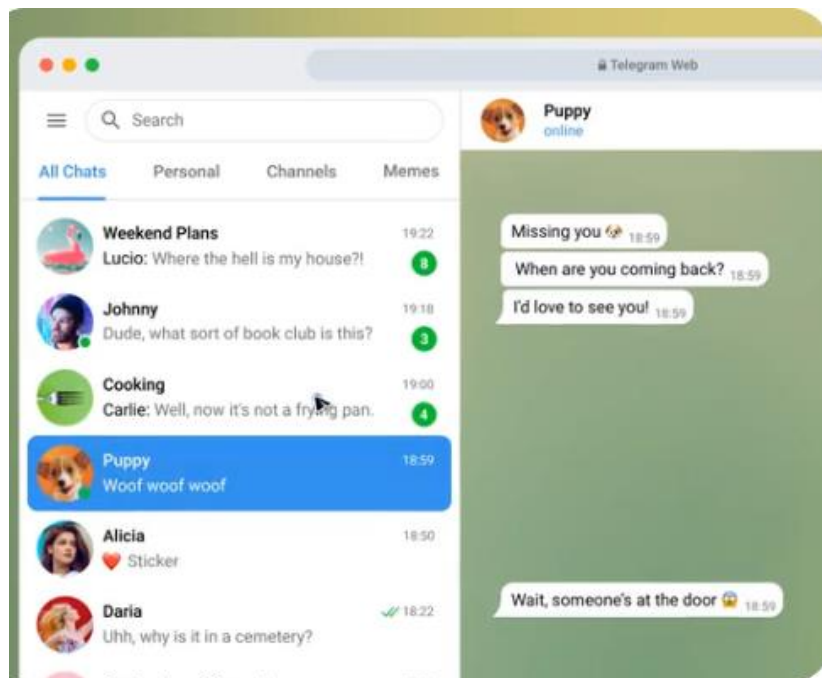


Рисунок 1.1 – Головна сторінка Telegram Web

Водночас, Telegram Web має певні недоліки. Можливості кастомізації інтерфейсу обмежені, а деякі функції, доступні в мобільній версії, такі як дзвінки та розширені можливості секретних чатів, у веб-версії відсутні або мають обмежений функціонал.

Іншим прикладом є WhatsApp Web [25]. На рис. 1.2 відображено інтерфейс, що синхронізується із мобільною версією WhatsApp.

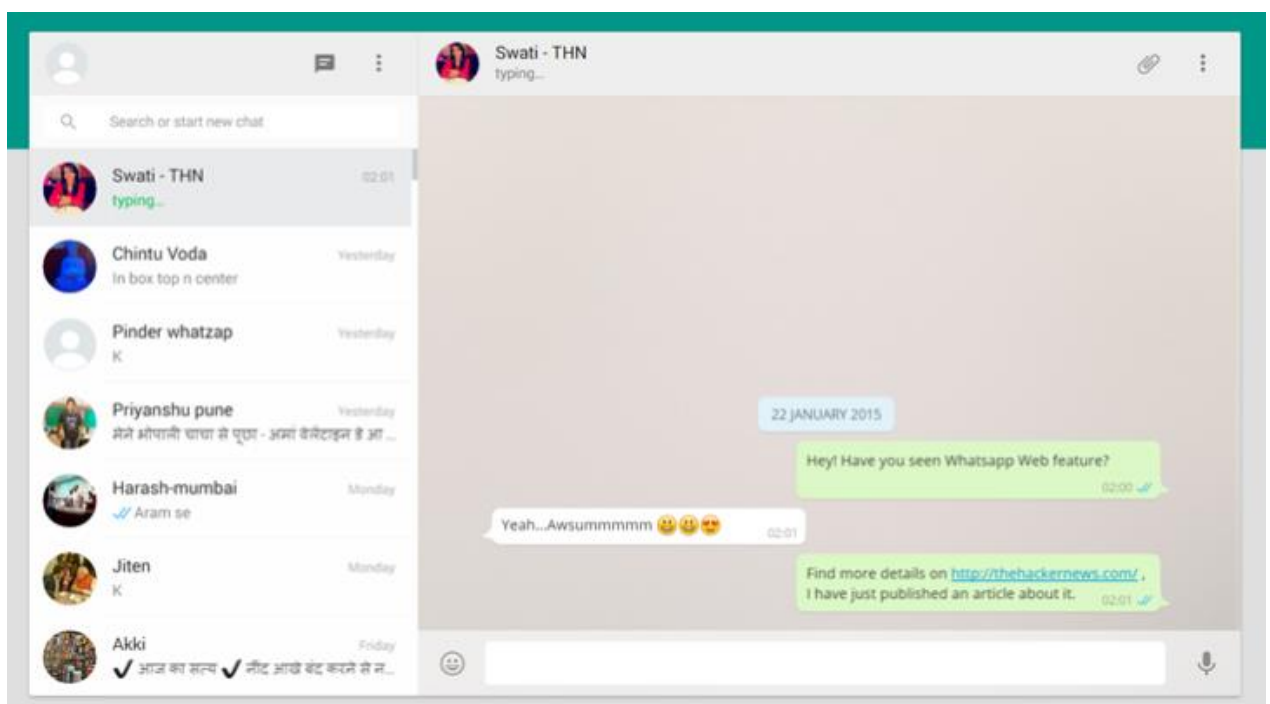


Рисунок 1.2 – Інтерфейс WhatsApp Web

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

WhatsApp Web дозволяє користувачам спілкуватися через браузер на комп'ютері, обмінюючись текстом, фото, аудіо та відео. Це зручно для тих, хто працює за комп'ютером і не хоче відволікатися на телефон. Веб-інтерфейс простий у використанні. Проте, WhatsApp Web вимагає постійного підключення до Інтернету на телефоні, що обмежує його автономність і може бути незручним, якщо телефон недоступний або без зв'язку.

Перейдемо до переваг WhatsApp Web. Цей вебмесенджер пропонує ряд значних переваг, що роблять його практичним рішенням для повсякденної комунікації. Його інтуїтивно зрозумілий інтерфейс забезпечує легкість у використанні та швидкий початок роботи. Синхронізація з мобільним додатком дозволяє користувачам мати доступ до актуальної інформації на різних пристроях. Безпека даних забезпечується завдяки end-to-end шифруванню, що гарантує конфіденційність особистої інформації.

Хоча WhatsApp Web і зручний, він не позбавлений недоліків. Головний з них – залежність від мобільного телефону: веб-версія працює лише тоді, коли телефон увімкнений та підключений до Інтернету. Якщо телефон вимкнеться або втратить зв'язок, WhatsApp Web також перестане працювати. Крім того, не можна створити новий обліковий запис безпосередньо у веб-версії – для цього потрібен мобільний застосунок. Нарешті, функціональність WhatsApp Web не відтворено повною мірою, що робить досвід користування менш насиченим.

На рис. 1.3 зображено головну сторінку вебверсії Facebook Messenger – одного з конкурентів, що дозволяє спілкуватися безпосередньо у браузері [11].

Популярність Facebook Messenger як платформи для онлайн-комунікації пояснюється низкою його переваг. Він дозволяє здійснювати відео- та аудіодзвінки без потреби завантажувати додаткові програми, які роблять його зручним для швидкого зв'язку безпосередньо з браузера. Facebook Messenger тісно інтегрований з іншими сервісами Facebook, такими як Marketplace, Instagram та ігри, створюючи єдиний простір для спілкування та розваг. Важливо, що він надійно працює у веб-версії, забезпечуючи доступ до основних функцій навіть без мобільного застосунку.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				13
Змн.	Арк.	№ докум.	Підпис	Дата		

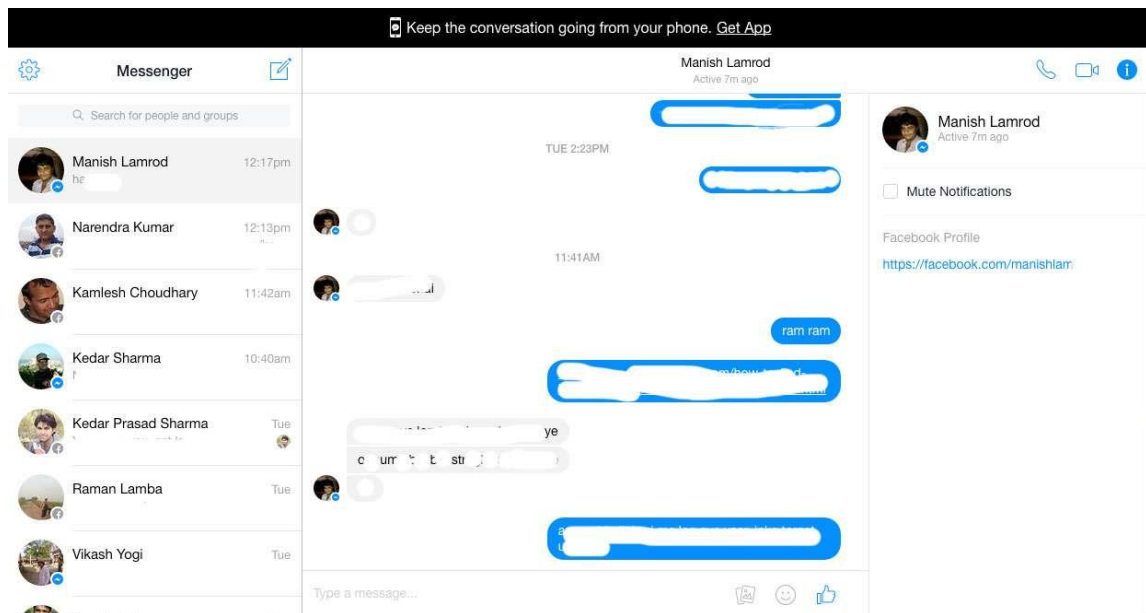


Рисунок 1.3 – Головна сторінка Facebook Messenger

Незважаючи на позитивні сторони, Facebook Messenger також має низку недоліків, які можуть знижувати якість користувацького досвіду. Обов'язкова наявність облікового запису Facebook може бути незручною для тих, хто не хоче активно користуватися соціальними мережами. Інтерфейс часто здається занадто завантаженим через велику кількість додаткових функцій та інтеграцій, що може ускладнювати навігацію та відволікати від спілкування. Крім того, наявність реклами, яка може з'являтися у вигляді повідомлень або блоків, погіршує загальний досвід користування. Хоча Messenger пропонує широкий спектр мультимедійних функцій, його залежність від облікового запису Facebook може бути обмежуючим фактором для деяких користувачів.

Оцінюючи інші популярні сервіси, варто звернути увагу на Slack [20] – ще одного конкурента, орієнтованого передусім на командну взаємодію. На рис. 1.4 представлено інтерфейс цієї платформи, яка активно використовується для спільної роботи, особливо в умовах віддаленої або гібридної зайнятості. Slack надає адміністраторам можливості для створення тематичних каналів, керування правами доступу та моніторингу активності користувачів. Завдяки інтеграціям із популярними сервісами, такими як Google Drive, Trello та Zoom, він слугує централізованим хабом для організації робочих процесів. Крім того, користувачі можуть обмінюватися файлами та автоматизувати рутинні завдання.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

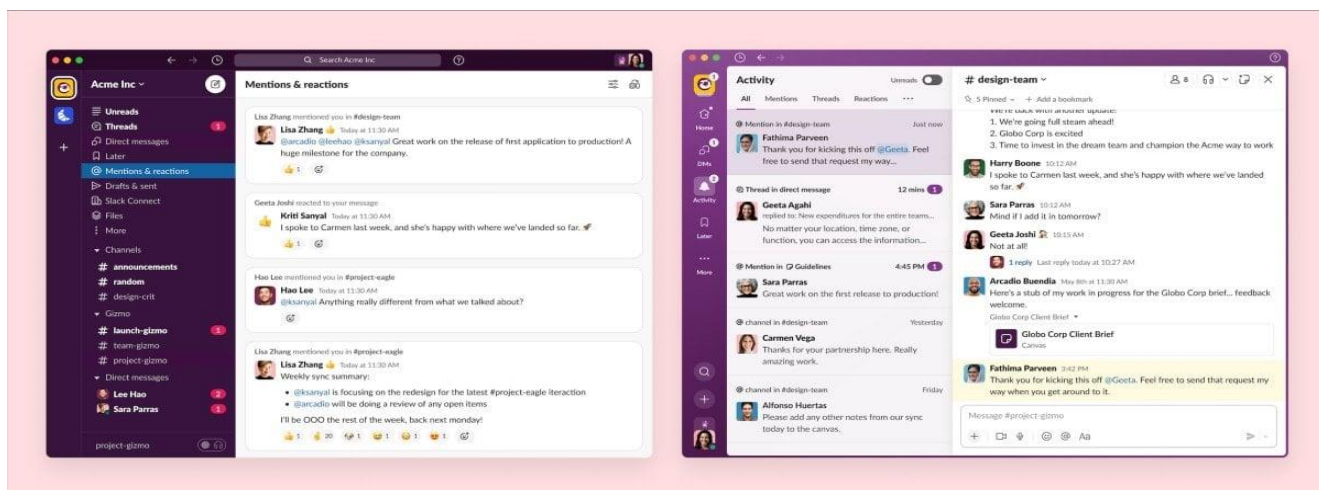


Рисунок 1.4 — Інтерфейс Slack

Розглядаючи переваги Slack, варто відзначити його зручне адміністрування та широкі можливості персоналізації, що дають змогу адаптувати платформу під конкретні потреби команди або компанії. Інтеграція з популярними інструментами, такими як Jira, GitHub і Google Docs, забезпечує ефективне управління завданнями та обмін інформацією без необхідності покидати єдиний робочий простір. Крім того, високий рівень безпеки та контроль доступу роблять Slack надійним рішенням для корпоративного використання.

Водночас, незважаючи на численні переваги, Slack має і певні недоліки. Зокрема, вартість преміум-підписки може бути досить високою, яка особливо відчутно для великих організацій, обмежуючи доступ до повного функціоналу. Для нових користувачів інтерфейс може здатися складним через велику кількість налаштувань і повідомлень. Також безкоштовна версія має обмеження на кількість збережених повідомлень і інтеграцій, яка може ускладнити роботу над довготривалими проектами.

Ще одним прикладом є Microsoft Teams [22], інтерфейс робочого простору якого наведено на рис. 1.5. Це універсальна платформа для командної роботи, котру широко використовують як у компаніях, так і в навчальних закладах. Microsoft Teams об'єднує інструменти для спілкування, організації роботи та спільної роботи над проектами. Користувачі можуть створювати групи та канали для обговорення конкретних тем, планувати зустрічі, а також проводити відеоконференції й онлайн-дзвінки – функції, які особливо актуальні для віддаленої роботи та навчання.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдяки інтеграції з сервісами Microsoft, такими як OneDrive, SharePoint і Office 365, команди мають змогу спільно редагувати документи в реальному часі, підвищуючи продуктивність та полегшуючи прийняття рішень. Окрім того, платформа надає потужні інструменти адміністрування для контролю доступу, забезпечення безпеки та управління даними в організації.

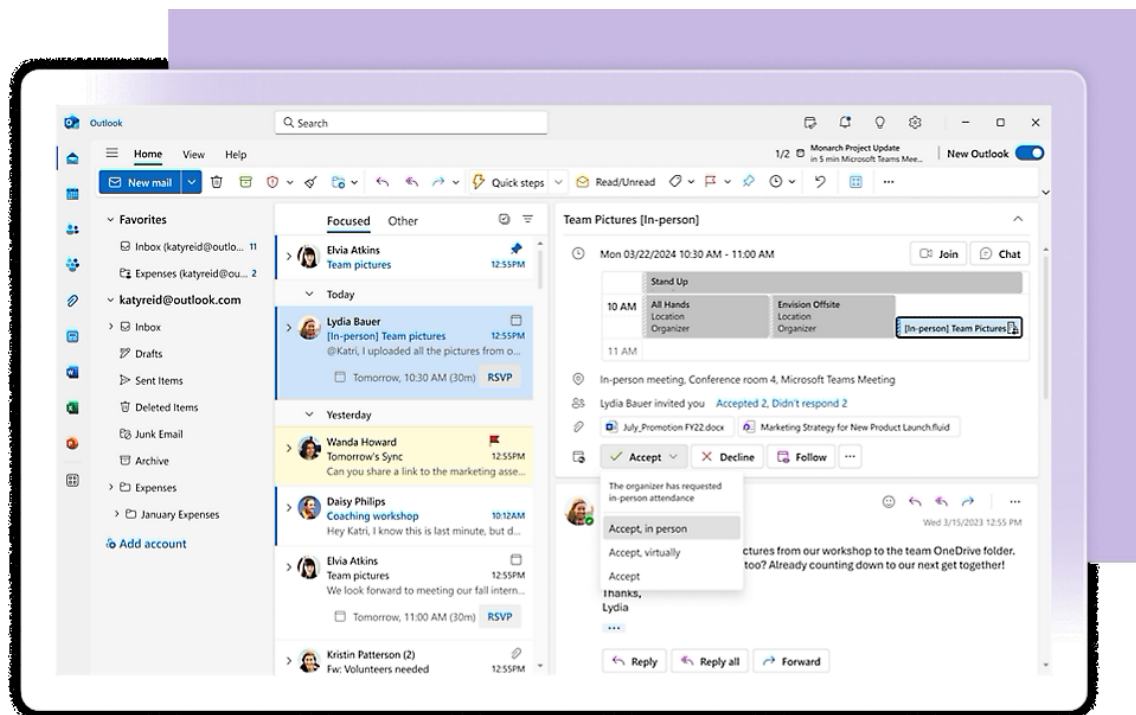


Рисунок 1.5 – Робочий простір Microsoft Teams

Розглядаючи переваги Microsoft Teams, варто виділити його розширені можливості адміністрування, які значно полегшують управління командами, налаштування прав доступу та оптимізацію робочих процесів у великих організаціях. Тісна інтеграція з Office 365 дозволяє користувачам легко співпрацювати над документами Word, Excel, PowerPoint та іншими файлами безпосередньо в середовищі Teams. Завдяки хмарній інфраструктурі Microsoft платформа забезпечує високу надійність, стабільність і безпеку, яка робить її привабливим вибором для бізнесу.

Проте, незважаючи на численні переваги, у Teams є й певні недоліки, які варто враховувати. Користувачі часто відзначають складність інтерфейсу, особливо ті, хто лише починає працювати з платформою або не має досвіду роботи з продуктами Microsoft. Обов'язкова наявність облікового запису Microsoft може

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

ускладнити залучення зовнішніх учасників. Крім того, Teams іноді суттєво навантажує систему, яка може призводити до уповільнення роботи комп'ютера, особливо на пристроях із обмеженими ресурсами або під час одночасного використання кількох застосунків.

Порівняльний аналіз оглянутих месенджерів зведений до табл. 1.1.

Таблиця 1.1

Таблиця порівняння аналогів вебмесенджерів

Характеристика	Telegram Web	WhatsApp Web	Messenger	Slack	Microsoft Teams
Підтримка мультимедіа	+	+	+	+	+
Робота без реєстрації номера телефону	+	-	+	+	+
Інтеграція з іншими сервісами	+/-	-	+	+	+
Наявність адміністративної панелі	-	-	-	+	+
Зручність інтерфейсу	+	+	+/-	+/-	+/-
Безкоштовний доступ	+	+	+	+/-	+/-

Виходячи з даних табл. 1.1, можна зробити висновок, що більшість сучасних месенджерів пропонують широкий спектр можливостей, включаючи мультимедійну підтримку та інтеграцію з різними сервісами. Проте, часто вони не відповідають потребам користувачів, які цінують простоту, незалежність від мобільних пристроїв та зручне адміністрування. Багато з них перевантажені

функціями, мають складний інтерфейс або вимагають встановлення мобільних застосунків. Тому існує потреба у створенні рішення, яке поєднує в собі доступність та легкість використання, не потребуючи додаткових інсталяцій та підключень.

Вебмесенджер “Connect” розроблений саме з урахуванням цих потреб. Він вирізнятиметься інтуїтивно зрозумілим інтерфейсом, зосередженим на зручному спілкуванні, та естетичним дизайном. “Connect” підтримуватиме як особисте, так і корпоративне використання, пропонуючи базові функції адміністрування для управління користувачами, їхніми правами доступу та обміном файлами. Це дозволить створити ефективне робоче середовище для команд, зберігаючи при цьому простоту та зручність для кожного користувача, без необхідності складних налаштувань або встановлення додаткового програмного забезпечення.

1.3. Вибір архітектури вебмесенджера

Архітектура вебмесенджера “Connect” спроектована з урахуванням вимог до гнучкості, масштабованості та надійності. Вона забезпечує ефективне управління обміном повідомленнями, автентифікацію користувачів, передачу файлів та адміністрування. Проектування передбачає можливість горизонтального масштабування та обробку великих обсягів даних, що необхідно для підтримки стабільної роботи при зростанні користувацької бази. Система реалізована за трирівневою архітектурою, що включає клієнтську частину, серверний компонент та базу даних, з чітким розподілом відповідальності для забезпечення високої продуктивності та надійності.

Об'єктно-орієнтоване програмування (ООП) було обрано як ключовий підхід для забезпечення високої якості та зручності підтримки коду месенджера. Завдяки ООП, основні компоненти, такі як користувачі, повідомлення, чати та файли, представлені у вигляді самостійних об'єктів з чітко визначеними властивостями та поведінкою. Це значно покращує структуру системи, роблячи її більш зрозумілою, гнучкою та адаптованою до майбутніх змін. Крім того, ООП дозволяє повторно використовувати код, що прискорює розробку. Наприклад, інтеграція нових типів

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

повідомлень, таких як аудіо- та відеофайли, може бути здійснена без необхідності переписувати основні частини програми. Саме тому об'єктно-орієнтований підхід забезпечує чистоту та зрозумілість коду. Крім того, як вже було зазначено, зміни в одному компоненті системи зазвичай не впливають на інші, які дозволяють уникати багатьох помилок при змінах. Завдяки цьому підходу, компоненти системи можна розробляти та вдосконалювати окремо один від одного. Це мінімізує ймовірність помилок, прискорює тестування та налагодження, а також спрощує додавання нових можливостей і розширення функціоналу без необхідності переробляти існуючі частини системи.

Для організації коду та забезпечення його модульності в архітектурі застосунку використано шаблон проєктування Model-View-Controller (MVC). Цей підхід розділяє застосунок на три основні частини: модель, представлення та контролер, що сприяє кращій організації коду, полегшує розробку, тестування та підтримку.

Модель (Model) відповідає за управління даними та бізнес-логіку. Вона визначає структуру даних (наприклад, користувачі, повідомлення, чати, медіафайли) та правила їх обробки. Модель також забезпечує доступ до бази даних, виконуючи операції збереження, оновлення та отримання даних.

Представлення (View) відповідає за відображення інформації користувачеві. У вебмесенджері “Connect” представлення реалізовано у вигляді веб-інтерфейсу, який розроблено таким чином, щоб бути інтуїтивно зрозумілим та зручним у використанні, дозволяючи навіть початківцям швидко з ним розібратися.

Контролер (Controller) виступає посередником між користувачем та моделлю. Він отримує запити від користувача, обробляє їх, викликає відповідні методи моделі для виконання необхідних дій, а потім передає результати у представлення для відображення. Такий поділ відповідальності дозволяє відокремити бізнес-логіку від інтерфейсу користувача, що значно спрощує тестування, внесення змін та подальший розвиток застосунку.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

1.4. Обґрунтування вибору інструментальних засобів

Для створення вебмесенджера “Connect” використовувалися сучасні, надійні та ефективні інструменти, які дозволили забезпечити гнучкість, масштабованість, безпеку та прискорений процес розробки.

Для розробки серверної частини вебмесенджера “Connect” використано NestJS, сучасний та продуктивний фреймворк на базі Node.js, який дозволяє створювати масштабовані серверні застосунки [12]. Його популярність зумовлена поєднанням високої продуктивності, гнучкості та зручності розробки. NestJS виділяється завдяки вбудованій підтримці TypeScript. Це дозволяє використовувати потужну систему типізації для зменшення кількості помилок на етапі розробки, роблячи код більш надійним і легким у підтримці, особливо в великих проектах. Також, можна додати, NestJS значно спрощує розробку REST API та WebSocket-з'єднань, саме це робить його найбільш підходящим вибором для месенджера. Швидка та стабільна обробка запитів і з'єднань – основа будь-якого месенджера, а NestJS забезпечує це. WebSocket дозволяє реалізувати двосторонній зв'язок у реальному часі, необхідний для миттєвого обміну повідомленнями, а REST API забезпечує легку інтеграцію з іншими сервісами та клієнтами [8]. Для вебмесенджера “Connect” потрібна серверна частина, яка була б швидкою в розробці, стабільною та легко масштабованою. NestJS відповідає цим вимогам, поєднуючи гнучкість Node.js з архітектурними рішеннями, подібними до Spring. Це дозволяє створити надійну та продуктивну серверну частину для “Connect” з мінімальними зусиллями.

Для підтвердження обґрунтованості вибору інструментів розробки було проведено детальний аналіз альтернативних фреймворків, які найбільш широко застосовуються для створення серверної логіки сучасних веб-застосунків. Враховувалися такі критерії, як продуктивність, масштабованість, підтримка безпеки, простота інтеграції з іншими сервісами, а також активність спільноти розробників і наявність документації. Це дозволило об'єктивно порівняти можливості різних рішень і вибрати оптимальний варіант, що найкраще відповідає вимогам проєкту. Детальні результати цього порівняння наведено в табл. 1.2.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				20
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця порівняння фреймворків

Фреймворк	Переваги	Недоліки
NestJS	Модульність	Вища складність в освоєнні початківцями
	Підтримка WebSocket	
	Інтеграція з ORM	
Express.js	Простий	Менше структурованості
	Швидкий старт	
	Велика спільнота	Складність масштабування
Spring Boot	Потужний, перевірений	Важчий стек
	Підтримка мікросервісів	Java-екосистема

Щодо клієнтської частини вебмесенджера “Connect”, вона реалізована за допомогою Next.js, потужного фреймворку, побудованого на основі React [13]. Вибір Next.js був зумовлений його здатністю створювати високопродуктивні, адаптивні та зручні веб-застосунки. Ключовою перевагою Next.js є підтримка серверного рендерингу (SSR), яка дозволяє значно оптимізувати швидкість завантаження сторінок. Завдяки SSR, початковий контент генерується на сервері, а не в браузері користувача, яка забезпечує швидке відображення інформації та покращує загальний досвід користування, який є критично важливим для месенджера, де швидкість обміну повідомленнями має першочергове значення [6]. Для стилізації інтерфейсу обрано Tailwind CSS, потужний інструмент, який значно прискорює розробку адаптивних та сучасних інтерфейсів [21]. Завдяки компонентному підходу Tailwind CSS, розробники можуть швидко створювати стилі, уникаючи складної CSS-архітектури. Це не тільки економить час, але й забезпечує повний контроль над зовнішнім виглядом, дозволяючи адаптувати інтерфейс до будь-якого екрану. У результаті використання Next.js дасть змогу створити адаптивний, інтуїтивно зрозумілий та високопродуктивний інтерфейс користувача. Автоматична генерація типізованих запитів забезпечує повну

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				21
Змн.	Арк.	№ докум.	Підпис	Дата		

інтеграцію з TypeScript, надаючи переваги статичної типізації та автодоповнення. Зручна система міграцій дозволяє легко вносити зміни до структури бази даних, автоматизуючи цей процес та мінімізуючи ризик помилок.

В якості основного сховища даних у проєкті обрана PostgreSQL – реляційна система управління базами даних з відкритим кодом, яка відома своєю високою продуктивністю, надійністю та стабільністю [16]. Такий вибір зумовлений широкими можливостями PostgreSQL щодо підтримки складних реляційних моделей даних, повноцінної реалізації ACID-транзакцій, що гарантує цілісність і консистентність інформації навіть за високого навантаження. Крім того, PostgreSQL характеризується великою гнучкістю в налаштуваннях, розширюваністю та можливістю масштабування, що особливо важливо для роботи з великими обсягами даних і системами з динамічно зростаючим навантаженням. Для інтеграції бази даних з серверною частиною, побудованою на фреймворку NestJS, використовується ORM-бібліотека TypeORM. Вона значно спрощує розробку, дозволяючи працювати з базою даних через об'єктно-реляційне відображення, забезпечує типізацію запитів і роботу з TypeScript, що підвищує безпеку коду і зменшує ризик помилок. Завдяки цьому підходу, реалізація бізнес-логіки стає більш зручною і ефективною, а взаємодія з базою даних є більш прозорою та керованою.

Для більш повного та всебічного обґрунтування вибору технологічних рішень у проєкті також було здійснено аналіз інших популярних систем управління базами даних, які широко використовуються у сучасних веб-застосунках. При цьому враховувалися ключові параметри, такі як продуктивність, надійність, підтримка масштабування, типи даних, які підтримують, рівень безпеки, а також зручність інтеграції з обраними фреймворками і середовищами розробки. Отримані порівняльні характеристики і особливості цих систем наведено в табл. 1.3, які дають змогу зробити обґрунтований вибір СУБД для реалізації проєкту. Цей аналіз допоміг визначити переваги та обмеження кожної з розглянутих систем, що є важливим для адаптації архітектури застосунка до конкретних бізнес-вимог.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

Таблиця порівняння баз даних

Назва	Тип	Переваги	Недоліки
MySQL	Реляційна	Підтримка складних зв'язків	Обмежені можливості для зберігання неструктурованих даних
		Висока цілісність даних	
		Проста в освоєнні	
		Широка документація	
PostgreSQL	Реляційна	Підтримка складних запитів	Складна в налаштуванні
		Підтримка великих даних та високої надійності	
MongoDB	NoSQL	Гнучкість для зберігання неструктурованих даних	Не підтримує складні транзакції
		Легке масштабування	Проблеми з цілісністю даних при великих навантаженнях і складних зв'язках між даними
		Висока швидкість для простих операцій	
SQLite	Реляційна	Простота	Висока вартість ліцензії
		Автономність	Не підходить для великих проєктів

Visual Studio Code (VS Code) є основним інструментом розробки цього проєкту. Цей швидкий та гнучкий редактор коду, з підтримкою розширень на кшталт ESLint, Prettier та GitLens, забезпечує комфортне та продуктивне середовище для веб-розробки. Інтелектуальні функції, дебагер та термінал, інтегровані у VS Code, дозволяють ефективно розробляти, тестувати та налагоджувати код.

Разом з розробкою коду, управління базою даних PostgreSQL відбувається за допомогою pgAdmin 4, офіційного графічного інструменту [15]. Він забезпечує візуальний доступ до створення та модифікації таблиць, виконання SQL-запитів, перегляду даних, моніторингу активних сесій та аналізу структури бази даних. Це дозволяє ефективно керувати схемами даних та відстежувати зміни в процесі розробки.

Окрім основного коду, для полегшення розробки застосовуються допоміжні інструменти. Наприклад, Postman допомагає перевіряти працездатність REST API, надсилаючи запити до серверної частини та контролюючи правильність відповідей, що критично важливо при виявленні та виправленні помилок [17]. Prisma Studio, у свою чергу, надає графічний інтерфейс для зручного перегляду та зміни даних у базі даних, що значно спрощує контроль за правильністю роботи з ORM [18]. Також, за необхідності, використовується Docker, який дозволяє запускати проєкт в ізольованому контейнері, гарантуючи стабільність конфігурації незалежно від операційної системи або налаштувань комп'ютера розробника [9].

Висновки до першого розділу

У першому розділі здійснено поглиблений огляд й аналіз існуючих аналогічних систем, їхніх можливостей, переваг і недоліків, з урахуванням актуальних потреб сучасних користувачів. На основі проведеного аналізу сформульовано вимоги до розроблюваного вебмесенджера, а також обґрунтовано вибір технологій та інструментів, що забезпечать ефективну розробку, підтримку, масштабування та можливість подальшого розширення функціоналу, а саме фреймворків NestJS, Next.js, базу даних PostgreSQL, WebSocket. TailwindCSS.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

Заплановано реалізацію особистих чатів для приватного спілкування, підтримку групових чатів із можливістю керування учасниками, функціонал адміністрування груп, включаючи призначення адміністраторів і налаштування прав доступу, можливість редагування профілю користувача, відстеження активності (статус онлайн, останній вхід, історія повідомлень), а також доступ до файлової структури для обміну файлами різних типів. Крім того, передбачено впровадження перспективного доповнення у вигляді навігації по файловій системі, що дозволить структурувати цифровий простір для ефективної командної взаємодії. Такий функціонал сприятиме створенню сучасного, безпечного і зручного засобу комунікації, придатного як для індивідуального, так і для колективного використання.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБМЕСЕНДЖЕРА

2.1. Визначення варіантів використання системи

У процесі реалізації обраного програмного рішення було створено власний вебмесенджер під назвою “Connect”. Це застосунок, побудований за клієнт-серверною архітектурною моделлю з використанням “тонкого” клієнта. Такий підхід передбачає, що основна обчислювальна логіка, обробка запитів, зберігання та управління даними зосереджені на стороні сервера, тоді як клієнт виконує переважно функції інтерфейсу для взаємодії користувача із системою.

З метою створення ефективного, безпечного та зручного інструменту для комунікації між користувачами було визначено низку функціональних вимог, які відображають очікувану поведінку та визначають її ключові можливості. Ці вимоги сформульовано відповідно до архітектури та цілей проєкту, з акцентом на забезпечення комфортної взаємодії, оперативного обміну інформацією та гнучкого керування обліковим записом.

Функціональні вимоги сформовані на основі цілей проєкту і охоплюють ключові можливості, необхідні для забезпечення зручної, ефективної та безпечної взаємодії між користувачами. В межах розробки передбачено реалізацію ряду функцій, серед яких, реєстрація та вхід до системи за допомогою номеру телефону з підтвердженням через SMS-код. Після авторизації користувач зможе змінювати свій профіль, зокрема оновлювати ім'я, завантажувати аватар. Спілкування здійснюється через чати, де можна створювати нові розмови, приєднуватися до існуючих, переглядати їх вміст, а також надсилати, редагувати й видаляти текстові повідомлення [5]. Крім того, підтримується обмін різноманітними файлами (зображення, документи тощо). Вся історія спілкування зберігається та доступна для перегляду. Обмін повідомленнями відбувається в режимі реального часу завдяки інтеграції технології WebSocket, яка забезпечує високу швидкість передачі даних.

Нефункціональні вимоги відіграють не менш важливу роль у створенні якісного вебмесенджера, адже вони визначають загальні характеристики системи,

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

які впливають на її стабільність, продуктивність і зручність використання. В межах проєкту було сформовано перелік таких вимог, які забезпечують надійність і масштабованість платформи. Однією з ключових характеристик є швидкість, тобто час між надсиланням та отриманням повідомлення не повинен перевищувати однієї секунди, оскільки це критично важливо для підтримки активної комунікації.

Системні вимоги проєкту визначають середовище розробки та розгортання вебмесенджера, які напряду впливають на його продуктивність, стабільність і масштабованість. Розробка серверної частини здійснюється в операційній системі Windows 10, яка забезпечує зручність програмування, тестування та налагодження коду, а також має широку підтримку сучасних інструментів розробки. Для продукційного середовища обрано операційну систему Linux, зокрема Ubuntu 22.04 LTS, яка відзначається високою надійністю, довготривалою підтримкою, активною спільнотою та сумісністю з більшістю сучасних серверних технологій. Такий підхід дозволяє забезпечити надійну та ефективну роботу системи в умовах високого навантаження.

Серверна частина вебмесенджера реалізована з використанням сучасних технологій, які забезпечують високу продуктивність і масштабованість системи. Основу становить платформа Node.js у поєднанні з прогресивним фреймворком NestJS, який дозволяє ефективно організувати архітектуру проєкту відповідно до принципів модульності та розділення відповідальностей. Збереження та обробка даних здійснюється в реляційній системі управління базами даних PostgreSQL, яка забезпечує надійність, цілісність і гнучкість роботи з інформацією. Для реалізації авторизації за номером телефону інтегровано SMS-шлюз Twilio, який дозволяє підтверджувати особу користувача через одноразові коди [24]. Миттєвий обмін повідомленнями між клієнтами забезпечується завдяки використанню WebSocket-сервера, побудованого на базі NestJS, який гарантує обробку даних у режимі реального часу.

Клієнтська частина системи реалізована з використанням сучасних вебтехнологій, які дозволяють забезпечити високу продуктивність, адаптивність інтерфейсу та зручність для кінцевого користувача. Фронтенд побудовано на

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				27
Змн.	Арк.	№ докум.	Підпис	Дата		

основі React 18 у поєднанні з фреймворком Next.js, що дає змогу створювати динамічні вебсторінки з підтримкою серверного рендерингу, маршрутизації та оптимізації продуктивності. Підтримуються всі сучасні операційні системи та браузері, включаючи Google Chrome, Microsoft Edge і Safari. Для безпечного обміну даними між клієнтом і сервером використовуються протоколи HTTPS та WebSocket, гарантуючи захист інформації та швидкий обмін повідомленнями в режимі реального часу.

Апаратні вимоги системи сформовані з урахуванням стабільної роботи як на стороні сервера, так і на клієнтських пристроях. Серверна частина потребує використання продуктивного обладнання, зокрема процесора з щонайменше чотирма ядрами, 8 ГБ оперативної пам'яті та SSD-накопичувача обсягом не менше 100 ГБ. Такі характеристики дозволяють системі ефективно обробляти великі обсяги запитів і забезпечувати безперебійну роботу навіть за умови високого навантаження. Водночас, для користувачів система залишається доступною навіть на пристроях із помірними ресурсами – достатньо 2 ГБ оперативної пам'яті та стабільного Інтернет-з'єднання для комфортної взаємодії з вебмесенджером у режимі реального часу.

Клієнтські застосунки, у свою чергу, слугують лише інтерфейсом для взаємодії користувача з сервером. Такий підхід забезпечує кросплатформеність, легке масштабування та підвищену безпеку даних. Клієнтська частина представлена веб-версією, розробленою на React, яка забезпечує динамічний та інтерактивний інтерфейс з оновленням даних в режимі реального часу. Серверна частина реалізована на NestJS. Це забезпечує високу продуктивність завдяки асинхронній моделі, також відповідає за маршрутизацію, авторизацію, обробку REST API запитів та взаємодію з базою даних. Це дозволяє легко масштабувати систему та забезпечувати стабільну роботу серверних процесів. Для зберігання даних використовується PostgreSQL – надійна та гнучка система управління базами даних, яка зберігає інформацію про користувачів, повідомлення, мультимедійні файли та інші службові дані. PostgreSQL підтримує складні транзакції та взаємозв'язки між даними. Обмін повідомленнями в реальному часі забезпечується

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

за допомогою WebSocket, який підтримує постійне з'єднання між клієнтом і сервером, дозволяючи миттєво надсилати та отримувати повідомлення. Для обробки мультимедійного контенту використовується сервіс Cloudinary, який відповідає за зберігання, обробку та доставку аватарок користувачів, що дозволяє знизити навантаження на сервер, тоді як інші файли обробляються та зберігаються безпосередньо через сервер у базі даних [7]. Загалом, архітектура вебмесенджера забезпечує легкість розгортання, високу продуктивність, безпеку даних та простоту масштабування, дозволяючи адаптувати систему до різних платформ та змін у функціоналі без значних змін у клієнтських застосунках [1].

Сценарії взаємодії користувачів системи показано у вигляді діаграми варіантів використання на рис. 2.1.

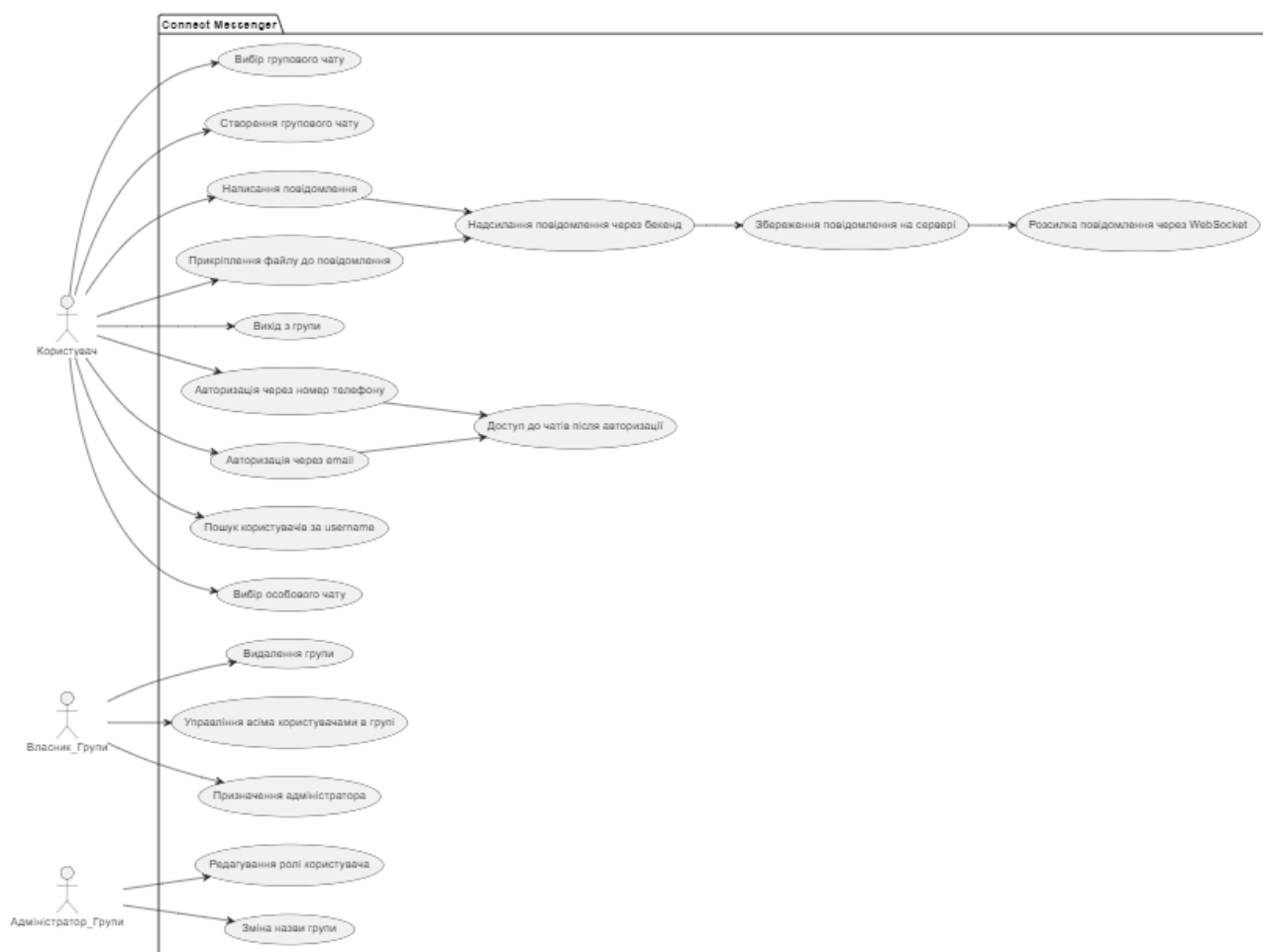


Рисунок 2.1 — Діаграма варіантів використання

Вхід та реєстрація в системі реалізовані з максимальною зручністю та надійністю. Користувач може легко отримати доступ до месенджера, обравши один із двох безпечних способів: за номером телефону з підтвердженням через SMS-код

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

або за e-mail з підтвердженням на пошту. Після успішної авторизації користувач одразу потрапляє до інтерфейсу чатів, готовий до спілкування. Ключовим елементом системи є функціонал обміну повідомленнями, який реалізований з використанням сучасних технологій. Користувач може ініціювати спілкування, вибравши існуючий чат або створивши новий. Він може вводити текстові повідомлення та додавати файли. Після натискання кнопки “Надіслати” система обробляє запит, повідомлення зберігається на сервері в базі даних, а потім, використовуючи WebSocket з’єднання, миттєво розсилається всім учасникам чату. Це забезпечує миттєву доставку повідомлень та їх відображення на пристроях користувачів в реальному часі, без необхідності ручного оновлення. Щоб поділитися файлом (мультимедійним або документом) у чаті, користувач вибирає потрібний чат, завантажує файл зі свого пристрою, і система автоматично відправляє його на сервер. Після завантаження, у чаті з’являється повідомлення з посиланням на цей файл, доступне для всіх учасників.

В межах проєкту було створено діаграму компонентів, яка ілюструє загальну структуру системи та взаємозв’язки між її ключовими складовими. Умовно її можна поділити на три логічні частини: клієнтську, серверну та інфраструктурну. Візуальне відображення цієї структури наведено на рис. 2.2.

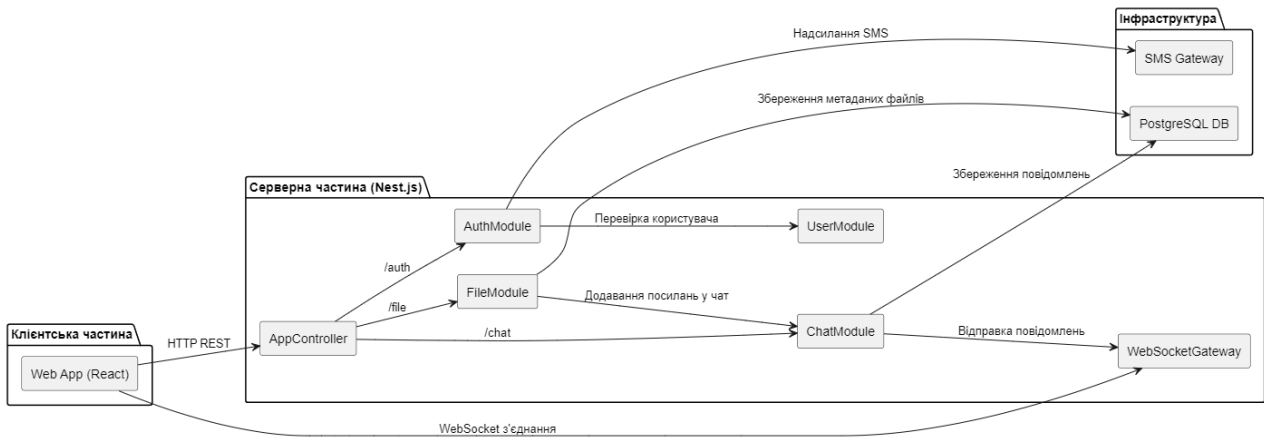


Рисунок 2.2 — Діаграма компонентів.

Фронтенд веб-версії, побудований на React, відповідає за відображення інформації та обробку дій користувача. Він комунікує з сервером за допомогою HTTP REST API, надсилаючи запити до AppController, а також підтримує двосторонній зв'язок у реальному часі через WebSocket-з'єднання з

WebSocketGateway. Серверна логіка реалізована за допомогою NestJS, фреймворку для Node.js, який дозволяє організувати код у вигляді незалежних модулів. AppController виконує роль “диспетчера”, обробляючи HTTP-запити, що надходять від клієнта, та розподіляючи їх між відповідними модулями для виконання необхідних операцій. Відповідальність за авторизацію та реєстрацію покладено на AuthModule. Він співпрацює з UserModule для перевірки наявності користувачів у системі та використовує SMS Gateway для надсилання SMS-повідомлень з метою підтвердження номерів телефонів. Модуль чату, ChatModule, відповідає за обробку повідомлень, їх збереження в базі даних PostgreSQL та розповсюдження в реальному часі серед усіх користувачів чату через WebSocketGateway. UserModule відповідає за безпечне управління користувацькими даними, зокрема за перевірку ідентичності користувачів, реєстрацію нових облікових записів та надання доступу через авторизацію. FileModule відповідає за обробку файлів: він приймає завантажені файли, зберігає інформацію про них в базі даних PostgreSQL, а потім надає посилання на ці файли для використання в ChatModule, зокрема для вставки в чат. Завдяки WebSocketGateway, реалізовано швидкий та ефективний обмін повідомленнями в чатах, де повідомлення миттєво передаються між користувачами через WebSocket-з'єднання. AppController виступає в ролі маршрутизатора, направляючи вхідні запити до відповідних функціональних модулів: аутентифікація обробляється AuthModule, чат – ChatModule, а робота з файлами – FileModule. AuthModule використовує SMS Gateway для надсилання SMS-повідомлень та UserModule для перевірки ідентичності користувачів. ChatModule відповідає за збереження історії чату в базі даних та розсилку повідомлень через WebSocketGateway. FileModule керує інформацією про файли та надає посилання на них для використання в ChatModule. Функціонування вебмесенджера забезпечується завдяки взаємодії між різними компонентами та зовнішніми сервісами. PostgreSQL DB служить центральним сховищем даних, де зберігається інформація про користувачів, їхні повідомлення та метадані файлів. Для надсилання SMS-повідомлень використовується SMS Gateway, з яким взаємодіє AuthModule. ChatModule і FileModule, в свою чергу, використовують PostgreSQL

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				31
Змн.	Арк.	№ докум.	Підпис	Дата		

DB для зберігання своїх даних. Створено веб-застосунок з використанням React для фронтенду та NestJS для бекенду. Комунікація між клієнтом та сервером реалізована через REST API та WebSocket. Серверна частина включає модулі для авторизації через SMS, управління чатами, обміну повідомленнями (текст та файли), адміністрування користувачів та реалізацію миттєвого обміну повідомленнями через WebSocket. Дані зберігаються в PostgreSQL, а для підтвердження номерів телефонів використовується SMS Gateway. Файли зберігаються в базі даних, а в чаті передаються посилання на них.

Оскільки діаграма компонентів надала загальне уявлення про архітектуру системи, наступним кроком стало детальне вивчення внутрішньої структури. Для цього була створена діаграма класів, яка показує, як організовані дані всередині системи, які атрибути мають об'єкти та як вони взаємодіють. Діаграма класів, представлена на рис. 2.3, є логічним продовженням попереднього етапу проєктування та дозволяє глибше зрозуміти внутрішню роботу системи. Вона наочно демонструє, які об'єкти використовуються у вебмесенджері, які дані вони містять та як взаємодіють між собою, формуючи цілісне уявлення про структуру даних і їхні зв'язки.

У вебмесенджері клас User відіграє критично важливу роль, забезпечуючи зберігання та обробку всієї інформації, пов'язаної з користувачем. Він включає в себе: унікальний ідентифікатор, номер телефону, який використовується для реєстрації та входу в систему, ім'я, що відображається в чатах, посилання на аватар, а також інформацію про поточний статус користувача (наприклад, “онлайн” або “не турбувати”). Клас User надає методи для аутентифікації користувача за допомогою коду авторизації та дозволяє користувачам оновлювати свій профіль, змінюючи ім'я або зображення.

Клас “Message” моделює структуру повідомлення, яке передається в чаті. Кожен об'єкт цього класу має унікальний ідентифікатор, містить дані про автора повідомлення, вказує на чат, до якого воно належить, зберігає текстовий вміст, фіксує час відправлення та визначає тип повідомлення (наприклад, текстове повідомлення або файл). Функціональність класу дозволяє відправляти, змінювати

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

та видаляти повідомлення.

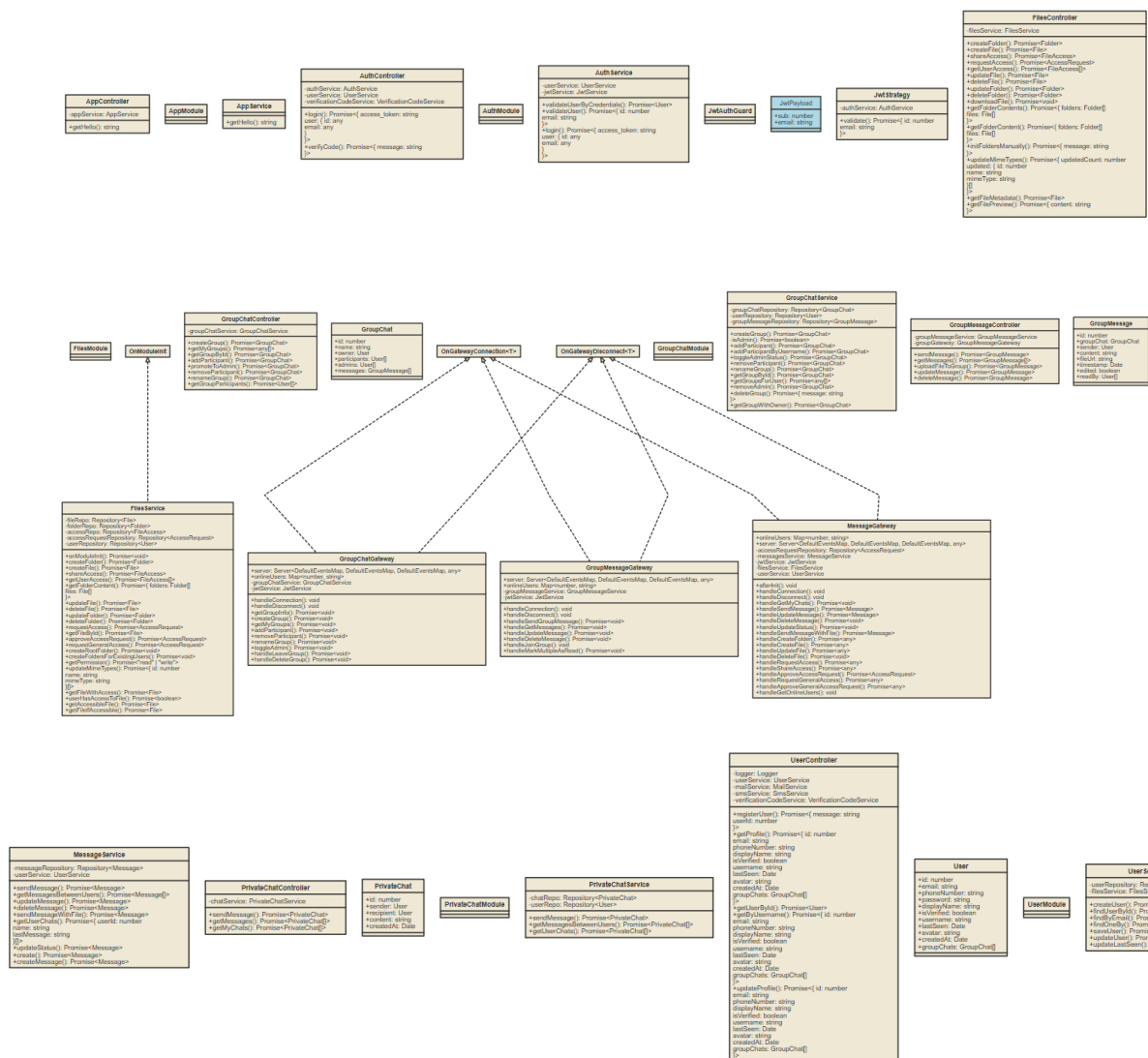


Рисунок 2.3 — Діаграма класів.

Клас “Message” моделює структуру повідомлення, яке передається в чаті. Кожен об'єкт цього класу має унікальний ідентифікатор, містить дані про автора повідомлення, вказує на чат, до якого воно належить, зберігає текстовий вміст, фіксує час відправлення та визначає тип повідомлення (наприклад, текстове повідомлення або файл). Функціональність класу дозволяє відправляти, змінювати та видаляти повідомлення.

Клас “Chat” призначений для управління даними, пов'язаними з груповими або приватними бесідами. Він містить унікальний ідентифікатор чату, перелік учасників, а також історію повідомлень, якими обмінювалися в рамках цього чату. Основні можливості класу включають додавання нових повідомлень до чату та

розширення списку учасників.

Для управління файлами, що передаються в чатах, використовується клас `File`. Він зберігає такі атрибути, як ID файлу, його розташування (URL), тип (наприклад, “зображення” або “документ”) та розмір у байтах. Клас також включає методи для завантаження файлів в систему та їх подальшого видалення.

Система побудована на основі чіткої логічної взаємодії між її складовими. Користувач має можливість ініціювати або відправляти значну кількість повідомлень. Кожен чат здатний вміщувати великий обсяг повідомлень та об'єднувати декількох учасників. Повідомлення, за потреби, можуть містити файлові вкладення. Завдяки такій архітектурі, система легко масштабується та адаптується до змін, що дозволяє безперешкодно додавати нові функції та повторно використовувати існуючі компоненти.

2.2. Розробка бази даних

Вся інформація, потрібна для функціонування месенджера, зберігається в спеціальній базі даних, яка забезпечує всі його можливості.

Архітектура бази даних вебмесенджера включає декілька ключових таблиць, кожна з яких відповідає за певний аспект функціональності. Серед основних: таблиці для зберігання інформації про користувачів (user), повідомлення (message), групові чати (group_chat), повідомлення в групах (group_message), учасників групових чатів (group_chat_participants_user) та адміністраторів груп (group_chat_admins_user).

Для створення цієї бази даних використовувалися SQL-запити, сумісні з PostgreSQL. Зокрема, таблиця “user” може бути створена таким чином:

```
CREATE TABLE public.user (  
    id SERIAL PRIMARY KEY,  
    email VARCHAR(255) UNIQUE NOT NULL,  
    phone_number VARCHAR(20) UNIQUE,  
    password_hash TEXT NOT NULL,  
    first_name VARCHAR(100),
```

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				34
Змн.	Арк.	№ докум.	Підпис	Дата		

```

last_name VARCHAR(100),
username VARCHAR(100) UNIQUE,
bio TEXT,
avatar_url TEXT,
is_verified BOOLEAN DEFAULT FALSE,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

Аналогічним чином створюються й інші таблиці, кожна з яких має свій набір полів, що відповідають за зберігання відповідної інформації.

Розглянемо детальніше діаграму бази даних, яка зображена на рис. 2.4.

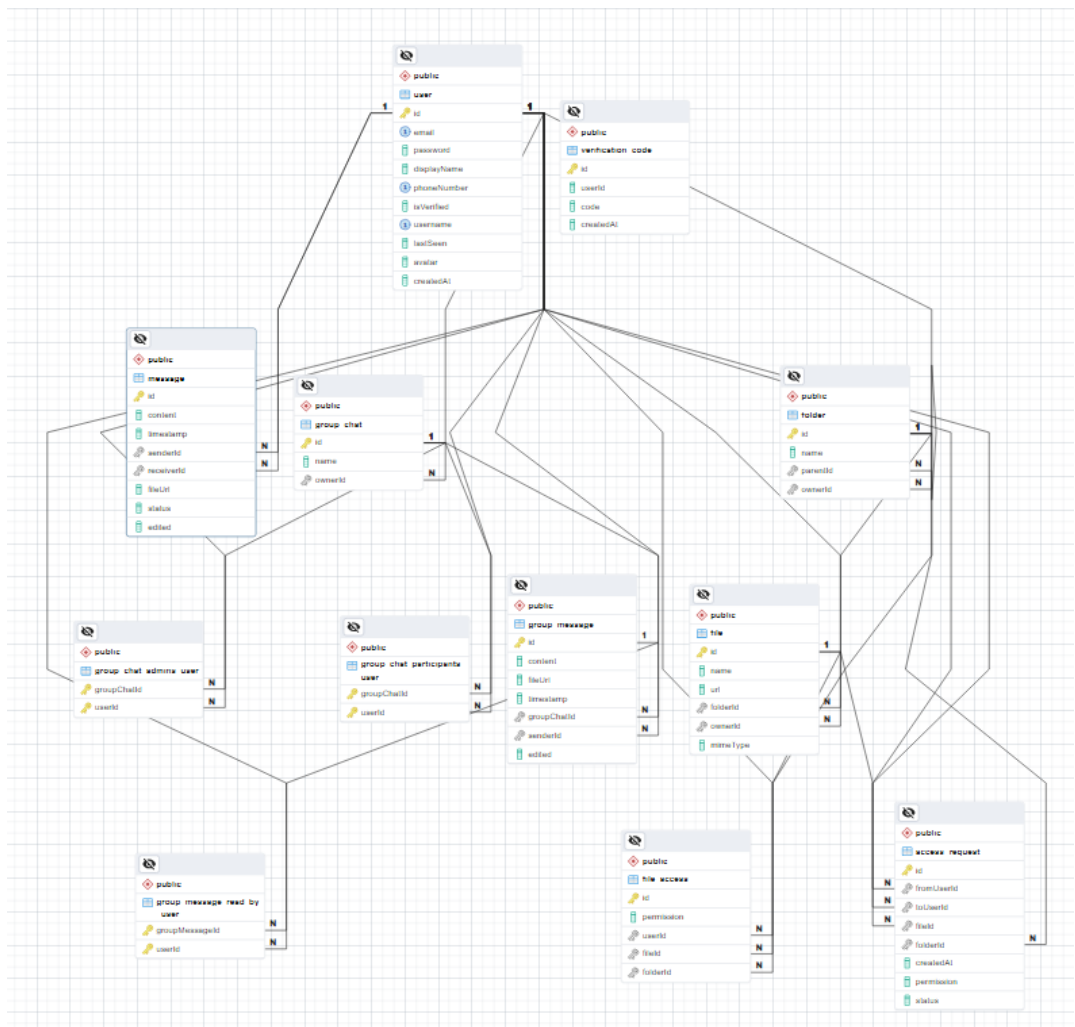


Рисунок 2.4 — Діаграма бази даних.

У структурі бази даних вебмесенджера передбачено 12 взаємопов'язаних таблиць, кожна з яких відповідає за зберігання та обробку специфічного типу інформації. Така побудова дозволяє забезпечити ефективну роботу системи,

включаючи автентифікацію користувачів, обмін повідомленнями, створення групових чатів, збереження файлів, управління правами доступу тощо. Кожна таблиця виконує окрему логічну функцію в межах загальної архітектури застосунку.

Детальний опис призначення кожної з таблиць наведено нижче. Розпочнемо з табл. 2.1, у якій подано структуру таблиці group_message, яка відповідає за зберігання повідомлень у групових чатах.

Таблиця 2.1

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор групового повідомлення
content	character varying	Текст повідомлення
fileId	integer	Ідентифікатор прикріпленого файлу (зв'язок з таблицею file)
timestamp	timestamp	Час відправлення повідомлення
groupChatId	integer	Ідентифікатор групового чату
userId	integer	Ідентифікатор користувача, який надіслав повідомлення
edited	boolean	Позначка, чи було повідомлення відредаговане

Наступною є таблиця group_chat, структура якої подана у табл. 2.2. Вона відповідає за зберігання інформації про групові чати, зокрема їх назву, власника та інші основні характеристики.

Таблиця 2.2

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор групового чату
name	character varying	Назва групового чату
ownerId	integer	Ідентифікатор користувача-власника чату

Продовжуємо розгляд структури бази даних. Таблиця group_chat_admin_user, подана у табл. 2.3, фіксує користувачів, які мають адміністративні повноваження в межах певного групового чату. Вона встановлює зв'язок між ідентифікатором чату та ідентифікатором користувача-адміністратора.

Таблиця 2.3

Поле	Тип	Призначення
groupChatId	integer	Ідентифікатор групового чату
userId	integer	Ідентифікатор користувача-адміна

Далі, таблиця group_chat_participants, представлена у табл. 2.4, описує участь користувачів у групових чатах. Вона реалізує зв'язок “багато-до-багатьох” між користувачами та чатами.

Таблиця 2.4

Поле	Тип	Призначення
groupChatId	integer	Ідентифікатор групового чату
userId	integer	Ідентифікатор учасника чату

Наступна таблиця group_message_read_by, представлена у табл. 2.5, вона зберігає інформацію про те, які користувачі вже прочитали певні повідомлення в групових чатах.

Таблиця 2.5

Поле	Тип	Призначення
groupMessageId	integer	Ідентифікатор повідомлення
userId	integer	Ідентифікатор користувача, який прочитав повідомлення

У табл. 2.6 наведено структуру таблиці message, яка використовується для зберігання особистих повідомлень між двома користувачами в системі веб-месенджера. Таблиця забезпечує збереження текстового вмісту повідомлення, ідентифікаторів відправника та отримувача для встановлення зв'язку між користувачами, а також статусу повідомлення (наприклад, “надіслано”,

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				38
Змн.	Арк.	№ докум.	Підпис	Дата		

“доставлено”, “прочитано”), яка дає змогу відслідковувати його стан у процесі спілкування. Крім того, у таблиці передбачене поле для збереження посилання на прикріплений файл (за наявності), яка дозволяє користувачам надсилати не лише текстові, а й мультимедійні повідомлення. Така структура сприяє повноцінній реалізації функціоналу обміну повідомленнями в межах платформи.

Таблиця 2.6

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор повідомлення
content	character varying	Текст повідомлення
senderId	integer	Ідентифікатор відправника
receiverId	integer	Ідентифікатор отримувача
fileId	integer	Ідентифікатор файлу
status	character varying	Статус повідомлення (прочитано/непрочитано)
edited	boolean	Позначка редагування повідомлення

Таблиця user, представлена у табл. 2.7, є ключовою у структурі бази даних, оскільки зберігає облікову інформацію про користувачів платформи: контактні, особисті дані, статус, фото профілю та дату створення облікового запису.

Таблиця 2.7

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор користувача
email	character varying	Email адреса користувача
password	character varying	Хеш паролю
displayName	character varying	Ім'я, яке відображається
phoneNumber	character varying	Номер телефону
photoUrl	character varying	Посилання на фото профілю
firstName	character varying	Ім'я
lastName	character varying	Прізвище
status	character varying	Статус користувача (наприклад, "онлайн")
createdAt	timestamp	Дата реєстрації

У табл. 2.8, подано структуру таблиці verification_code, яка відповідає за зберігання одноразових кодів підтвердження під час реєстрації чи авторизації користувача.

Таблиця 2.8

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор коду
userId	integer	Ідентифікатор користувача
code	integer	Код підтвердження
createdAt	timestamp	Час створення коду

Таблиця folder, представлена у табл. 2.9, використовується для створення та організації структури папок користувачів у файловій системі месенджера.

Таблиця 2.9

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор папки
name	character varying	Назва папки
parentId	integer	Ідентифікатор батьківської папки
ownerId	integer	Ідентифікатор власника (user)

Наступною є таблиця file_access, структура якої подана у табл. 2.10. Вона використовується для керування правами доступу до файлів. Вказує, який користувач має доступ до якого файлу, і з якими правами. Це дає змогу реалізувати

гнучку систему розмежування доступу, забезпечуючи як особисте користування файлами, так і їхнє спільне використання з іншими користувачами.

Таблиця 2.10

Поле	Тип	Призначення
fileId	integer	Ідентифікатор файлу
userId	integer	Ідентифікатор користувача, який має доступ
permission	text	Тип доступу (наприклад, "читання", "редагування")

Табл. 2.11 демонструє структуру таблиці file, яка зберігає інформацію про всі файли в системі: назву, тип, розмір, належність до папки та користувача-власника.

Таблиця 2.11

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор файлу
name	character varying	Назва файлу
folderId	integer	Ідентифікатор папки, до якої належить файл
type	character varying	MIME-тип файлу
size	integer	Розмір файлу
ownerId	integer	Ідентифікатор користувача-власника

Завершує огляд таблиця access_request, представлена у табл. 2.12. Вона фіксує запити на доступ до файлів між користувачами, зберігаючи інформацію про ініціатора, отримувача, статус та дату створення запиту.

Таблиця 2.12

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор запиту
fromUserId	integer	Користувач, який зробив запит
toUserId	integer	Користувач, до якого звертаються
fileId	integer	Ідентифікатор файлу
status	character varying	Статус запиту (очікує, дозволено, відхилено)
createdAt	timestamp	Дата створення запиту

2.3. Розробка функціональних алгоритмів роботи програми

Для забезпечення неперервної роботи вебмесенджера розроблено ряд алгоритмів, що керують ключовими процесами: реєстрацією та авторизацією користувачів, обміном повідомленнями (особистими та груповими), створенням та адмініструванням групових чатів. Спочатку відбувається реєстрація, користувач заповнює форму, вказуючи необхідні дані (email, пароль, ім'я тощо). Система перевіряє унікальність email та імені користувача. Пароль надійно шифрується за допомогою шифрування паролей. Після успішної перевірки створюється новий обліковий запис користувача. Також, якщо користувач уже був зареєстрований в системі, тобто його дані уже є в базі даних, тоді він може пройти авторизацію. Для цього, користувач вводить свій email або номер телефону та пароль. Система шукає відповідний обліковий запис і перевіряє пароль, порівнюючи його хеш. У разі успіху генерується токен доступу JWT, який використовується для підтвердження

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				43
Змн.	Арк.	№ докум.	Підпис	Дата		

прав користувача при подальших запитах [10]. У вебмесенджері будуть реалізовані як приватні так і групові чати. Що до приватних, то користувач пише повідомлення та вказує одержувача. Система перевіряє, чи існує такий користувач. Потім створюється запис про повідомлення, яке містить інформацію про відправника, одержувача, текст, вкладення (якщо є) та час відправлення. Завдяки WebSocket технології, повідомлення миттєво доставляється одержувачу без необхідності оновлювати сторінку. Тепер, після того як описали обмін приватними повідомленнями, можемо перейти до групових повідомлень, тобто до групових чатів. Для того щоб розпочати комунікацію, користувач створює групу, задаючи назву, та (за бажанням) додає користувачів. Система створює запис про групу, автоматично додає користувача як учасника та адміністратора. Після цього до групи можна запрошувати інших користувачів. Тепер, коли зрозуміло зі створенням, перейдемо до того, як працюють групові повідомлення. Користувач пише повідомлення в групі. Система перевіряє, чи є він учасником цієї групи. Якщо так, створюється запис про повідомлення, яка містить текст, інформацію про відправника, ідентифікатор чату та час відправлення. Повідомлення миттєво розсилається всім учасникам групи через WebSocket. Також у групах є керування, а саме користувач який створює групу він автоматично стає адміністратором групи, після цього у нього є можливість наприклад, змінити назву і можливість додати нових користувачів. Тепер перейдемо до того як призначати нових користувачів у групі. Поточний адміністратор призначає іншого учасника адміністратором. Система перевіряє права поточного адміністратора та надає новому адміністратору відповідні права. Що до додавання учасників в ролі адміністратора, адміністратор(користувач) запрошує користувача до групи. Система перевіряє права адміністратора та додає нового користувача до списку учасників. Новий учасник отримує сповіщення про те що він був доданий до групи в ролі адністратора.

Серверна частина розроблена з використанням API-ендпоінтів для кожного алгоритму, що забезпечує чітку структуру та ефективну обробку запитів. Інтеграція з WebSocket дозволяє миттєво передавати зміни на клієнтську сторону,

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

забезпечуючи користувачам досвід роботи в реальному часі без затримок. Стабільне WebSocket-з'єднання мінімізує навантаження на мережу. Безпека гарантується завдяки обов'язковій авторизації та валідації даних на кожному API-ендпоінті [14]. В результаті, користувачі отримують стабільну, швидку та зручну систему.

2.4. Реалізація вебмесенджера

У межах цього проєкту було реалізовано повноцінний вебмесенджер нового покоління, орієнтований на забезпечення максимально зручного, безпечного та функціонально насиченого середовища для онлайн-комунікацій. Платформа поєднує інтуїтивний інтерфейс із широким спектром можливостей, яка робить її придатною як для особистого спілкування, так і для організації взаємодії всередині команд чи спільнот. Основний акцент зроблено на стабільній роботі в режимі реального часу – усі повідомлення, зміни в чатах або передача контенту відбуваються миттєво завдяки використанню WebSocket-технології (додаток А), це забезпечує безперервне з'єднання між клієнтом і сервером. Користувачі можуть обмінюватися як текстовими, так і мультимедійними повідомленнями, включно з фотографіями, документами, відео та іншими типами файлів. Для кожного надісланого файлу реалізовано систему попереднього перегляду прямо у вікні чату (додаток Б), яка дозволяє ознайомитися з вмістом, не завантажуючи його на пристрій. У разі потреби, користувач завжди може зберегти файл локально. Завдяки інтеграції з системою зберігання даних, усі вкладення надійно зберігаються у файловій структурі, яка прив'язана до конкретного користувача та чату, з можливістю швидкого доступу до архіву навіть через тривалий час після надсилання.

Серед розширених функцій – повноцінне адміністрування групових чатів. Власники груп мають можливість призначати адміністраторів, обмежувати права учасників, видаляти повідомлення інших користувачів або організовувати доступ до спільних ресурсів. Це створює гнучку систему керування, що підходить як для звичайних обговорень, так і для організаційної діяльності в рамках колективів.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

Окрему увагу в процесі розробки було зосереджено на інтеграції додаткових сервісів, що суттєво розширюють межі традиційного обміну повідомленнями, перетворюючи звичайний чат на універсальну платформу для комунікації, взаємодії та вирішення завдань. Серед найбільш інноваційних рішень варто відзначити впровадження інтерактивного чату з вбудованим штучним інтелектом, здатного не лише підтримувати діалог у зручній формі, а й активно допомагати користувачам у широкому спектрі питань. Такий інтелектуальний співрозмовник може відповідати на запити, пояснювати складні теми, надавати уточнення, шукати релевантну інформацію в режимі реального часу, а також виконувати математичні розрахунки чи генерувати тексти, поради, плани, ідеї або інший корисний контент на основі потреб користувача. Завдяки цьому чат із помічником на базі штучного інтелекту стає незамінним інструментом у різних сферах — від освіти, де він може допомагати учням та студентам із виконанням завдань і поясненням матеріалу, до професійного середовища, де співробітники можуть швидко отримувати аналітичну або довідкову інформацію, оптимізуючи свою щоденну роботу. Така функціональність не лише підвищує ефективність спілкування, а й створює нові можливості для продуктивної взаємодії, заощаджуючи час, зменшуючи кількість рутинних дій і сприяючи прийняттю обґрунтованих рішень на основі швидко отриманої інформації.

Крім того, у систему впроваджено функцію спільного перегляду відео, що забезпечує можливість одночасного перегляду одного й того самого відеозапису кількома користувачами з повною синхронізацією відтворення у режимі реального часу. Завдяки цьому функціоналу учасники можуть бачити однакові кадри в однакову мить, що створює ефект спільної присутності та сприяє інтерактивному спілкуванню під час перегляду. Така можливість відкриває нові горизонти для організації колективного дозвілля, дозволяючи друзям або членам родини разом насолоджуватися фільмами, серіалами чи відеокліпами, незалежно від їхнього фізичного місцезнаходження. Окрім розважального аспекту, ця функція є надзвичайно корисною у сфері дистанційного навчання, де викладач може демонструвати навчальні відеоматеріали студентам із гарантією того, що всі

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

переглядають контент синхронно, що, у свою чергу, підвищує ефективність засвоєння матеріалу. Також це дає змогу проводити презентації мультимедійного контенту під час відеоконференцій або онлайн-зустрічей, коли важливо, щоб усі учасники бачили демонстраційні матеріали одночасно, що значно покращує якість візуального сприйняття інформації та загальну координацію під час обговорення.

Проект охоплює не лише базову функціональність месенджера, а й розробку пов'язаних модулів: система автентифікації з розмежуванням прав доступу, механізми захищеного зберігання та обміну файлами, логіка адміністрування чатів і груп, інтерфейс управління користувачською файловою структурою. Усе це реалізовано з урахуванням масштабованості архітектури, що дозволяє в подальшому адаптувати платформу під більшу кількість користувачів або інтегрувати нові функції без необхідності суттєвої перебудови системи.

Фундаментальною основою вебмесенджера є система двостороннього обміну повідомленнями, що працює на базі технології WebSocket. Завдяки цій технології забезпечується стабільне з'єднання між клієнтом і сервером, яке дозволяє передавати дані миттєво, без необхідності перезавантаження сторінки або періодичного опитування сервера. Це створює враження безперервного, живого спілкування та значно підвищує загальну швидкість реакції інтерфейсу, що, в свою чергу, робить досвід взаємодії з системою природним і динамічним.

Механізм обміну повідомленнями не обмежується лише їх надсиланням. Користувачі мають можливість редагувати раніше надіслані повідомлення або повністю їх видаляти, що особливо корисно у випадках, коли необхідно скоригувати помилку або видалити інформацію, яка більше не є актуальною. Така гнучкість управління контентом дозволяє користувачам зберігати контроль над комунікацією та підтримувати її актуальність. Крім того, реалізована функція індикації прочитання повідомлень, що дозволяє відстежити, коли саме співрозмовник ознайомився з інформацією. Це сприяє кращому розумінню контексту діалогу, особливо в ситуаціях, коли очікується відповідь або реакція.

Інтерфейс месенджера виконаний у мінімалістичному стилі, що відповідає сучасним стандартам дизайну цифрових продуктів (додаток В). Основна увага

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

приділяється чистоті візуального оформлення, типографіці, контрастності елементів та чіткому візуальному поділу функціональних зон. Такий підхід забезпечує не лише естетичну привабливість, а й інтуїтивну зрозумілість інтерфейсу, роблячи платформу зручною для користувачів з різним рівнем технічної підготовки. Незалежно від того, чи використовується вебмесенджер на смартфоні, планшеті чи комп'ютері, усі елементи коректно масштабуються та адаптуються під конкретний екран, що гарантує комфортне використання на будь-якому пристрої.

Таким чином, система двостороннього чату в поєднанні з гнучким керуванням повідомленнями, індикаторами статусу та ретельно опрацьованим інтерфейсом створює надійну, швидку та зручну платформу для сучасного цифрового спілкування. Вона задовольняє як базові потреби у щоденній комунікації, так і вимоги до професійного використання в динамічному середовищі командної взаємодії.

Групові чати реалізовані з особливою ретельністю та увагою до потреб користувачів, аби створити максимально комфортне середовище для колективного спілкування та обміну інформацією (додаток Г). Створення нової групи відбувається інтуїтивно просто, що дозволяє кожному користувачеві швидко ініціювати спільне обговорення на будь-яку тему або для будь-якої мети – від навчальних проєктів до неформального спілкування з друзями чи колегами. Користувачі мають змогу не лише створювати групи, а й приєднуватися до вже існуючих, залишаючи групу у будь-який момент за власним бажанням, що забезпечує свободу участі у спільнотах. В межах кожної групи доступна повноцінна взаємодія, що включає обмін текстовими повідомленнями, медіафайлами, посиланнями та іншими матеріалами, а також можливість швидкого реагування на повідомлення інших учасників, що робить спілкування живим та динамічним. Важливою особливістю є наявність розширеного адміністративного функціоналу, завдяки якому адміністратори груп мають необхідні інструменти для організації та підтримання порядку в чаті. Зокрема, вони можуть змінювати адміністратора, що дозволяє передавати права керування іншим учасникам у разі

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

потреби, додавати нових користувачів до групи або, навпаки, видаляти тих, хто більше не бере участі в обговореннях, редагувати назву групового чату відповідно до зміни його тематики чи цілей, а також переглядати повний список учасників для ефективного управління складом спільноти. Усі ці можливості створюють гнучку, добре керовану структуру групової взаємодії, яка відповідає сучасним вимогам до онлайн-комунікації та сприяє побудові активних, організованих і цілеспрямованих групових дискусій

Окрім обміну текстовими повідомленнями в режимі реального часу, вебмесенджер забезпечує повноцінну підтримку передачі файлів між користувачами. Система дозволяє надсилати та отримувати файли різних форматів, включаючи зображення, документи, відео, архіви та інші типи вкладень відповідно до встановлених політик безпеки. Такий підхід надає користувачам гнучкі можливості як для особистого спілкування, так і для професійного використання, наприклад, обміну робочими файлами або презентаціями без необхідності покидати інтерфейс платформи. Передані файли автоматично інтегруються в систему зберігання даних месенджера. Це дозволяє логічно прив'язати вкладення до конкретних повідомлень, зберігати метадані, такі як час відправлення чи автор, а також забезпечити доступ до матеріалів у зручному структурованому вигляді. Користувач може повернутися до будь-якого надісланого або отриманого файлу навіть через тривалий час, оскільки дані надійно зберігаються на сервері без дублювання або втрати інформації. Робота з файлами максимально спрощена завдяки інтегрованому перегляду та можливості завантаження безпосередньо з інтерфейсу месенджера. Зображення, документи та інші підтримувані формати можна переглядати прямо у вікні чату, а при потребі – зберегти на пристрій. Для зручності користувача файли можуть бути попередньо оброблені системою, наприклад, автоматично стискатися або генерувати мініатюри для швидкого перегляду. Це дозволяє оптимізувати продуктивність платформи навіть у випадках передачі великих об'ємів даних. Значна увага приділяється питанню безпеки, передача файлів відбувається через зашифровані канали зв'язку, а доступ до вкладень мають лише уповноважені користувачі (додаток Г). У групових чатах

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова, Т. М.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

реалізовано контроль прав доступу до кожного окремого файлу, який дозволяє уникнути несанкціонованого доступу або перегляду. Завдяки таким механізмам користувачі можуть бути впевнені у конфіденційності свого контенту, що особливо важливо при обміні чутливою або приватною інформацією. Функціональність передачі файлів є кросплатформною – вона однаково ефективно працює як у десктопних браузерях, так і на мобільних пристроях з різними операційними системами. Адаптація до умов повільного Інтернет-з'єднання, підтримка кешування, а за потреби й інтеграція з CDN-інфраструктурою забезпечують високу швидкість завантаження та зменшують затримки під час доступу до файлів. Тому, система передачі файлів у межах вебмесенджера не лише доповнює базову комунікаційну функцію, а й формує основу для повноцінної цифрової взаємодії. Вона сприяє продуктивній роботі команд, покращує зручність користувацького досвіду і відкриває нові сценарії використання платформи – від повсякденного обміну медіафайлами до професійної спільної роботи з документами та проектами.

Реалізовано повноцінний та функціонально багатий окремий модуль для забезпечення віртуального доступу до файлів користувача (додаток Д). Цей модуль відіграє ключову роль у розширенні можливостей спільної роботи та обміну даними між учасниками чату, дозволяючи користувачам надавати іншим особам контрольований і вибірковий доступ до своєї файлової структури. Йдеться не просто про передачу окремих файлів, а про створення динамічного середовища, у якому власник файлів самостійно визначає, які конкретні папки або документи можуть бути переглянуті іншими, що значно підвищує безпеку та персоналізованість процесу.

Вся взаємодія з системою здійснюється через зручний та інтуїтивно зрозумілий веб-інтерфейс, у межах якого користувачі, які отримали відповідний доступ, можуть переглядати вміст наданих папок без потреби у встановленні додаткового програмного забезпечення або здійсненні складних дій. Інтерфейс надає можливість перегляду структури файлів у вигляді знайомої ієрархії – подібно до локального файлового менеджера, що дозволяє швидко орієнтуватися у вмісті навіть великих директорій. Важливим аспектом є те, що користувачі можуть не

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

просто переглядати, а й завантажувати окремі файли, які їм потрібні, без необхідності завантаження повних архівів або додаткових конвертацій, що, у свою чергу, суттєво економить час, трафік і обчислювальні ресурси.

Механізм доступу передбачає як тимчасову, так і постійну форму відкриття доступу – залежно від цілей і потреб користувача, який надає доступ. Це означає, що один і той самий набір файлів може бути доступний протягом певного часу або постійно, з можливістю в будь-який момент змінити права доступу або повністю закрити доступ до файлів. Особлива увага приділена безпеці такої взаємодії: усі запити до файлів проходять через систему аутентифікації та авторизації, яка перевіряє, чи має конкретний користувач право переглядати і завантажувати вказані матеріали. Це унеможливорює випадкове або несанкціоноване розповсюдження приватної інформації та дозволяє зберігати повний контроль над власними цифровими ресурсами.

Модуль також враховує потреби в обробці великих обсягів інформації, що особливо актуально у випадках командної роботи над проектами з великою кількістю документів, зображень, відео або технічної документації (додаток Е). Система оптимізована таким чином, щоб забезпечувати високу швидкість завантаження та стабільний доступ навіть при значному навантаженні, з можливістю інтеграції з хмарними сервісами та локальними файловими сховищами. Такий підхід дозволяє ефективно організовувати обмін матеріалами в рамках групових чатів або індивідуального листування, створюючи умови для продуктивної віддаленої співпраці без потреби у зовнішніх інструментах. У підсумку, даний модуль є сучасним, безпечним і високоефективним рішенням, яке перетворює вебмесенджер на універсальний засіб не лише для комунікації, а й для управління файлами та спільної роботи над цифровим контентом.

Висновки до другого розділу

У другому розділі було детально викладено алгоритм функціонування веборієнтованої системи обміну повідомленнями – вебмесенджера. Розроблено структуру бази даних PostgreSQL, яка складається з дванадцяти таблиць, які

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечують зберігання інформації про користувачів, повідомлення, чати, реальні підключення та інші важливі елементи системи. Особлива увага була приділена створенню діаграми класів, що ілюструє основні об'єкти месенджера, їх атрибути і зв'язки, а також ролі кожного класу в архітектурі проекту.

Крім того, проведено детальний аналіз основних класів і компонентів системи, описані методи та функції, які реалізовані в їх межах, з зазначенням їх призначення та місця в загальній логіці роботи вебмесенджера. Також були розглянуті принципи обробки повідомлень у реальному часі за допомогою WebSocket-з'єднань, механізм автентифікації користувачів, обробка медіафайлів, а також реалізація основних функцій обміну текстовими повідомленнями та зображеннями.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБМЕСЕНДЖЕРОМ

3.1 Структура інтерфейсу вебмесенджера

Заходячи на вебмесенджер, користувачу відкривається головна сторінка, на якій розміщені авторизація та реєстрація користувача. Якщо користувач уже є зареєстрований в системі то він проходить авторизацію, що зображено на рис. 3.1, вводячи свої реєстраційні дані. В іншому випадку, коли користувач не має свого аккаунту, то йому потрібно пройти процедуру реєстрації, як видно на рис. 3.2. Для цього йому потрібно ввести або номер телефону або адресу електронної пошти, також потрібно ввести його унікальне ім'я і відображуване ім'я, перше буде використовуватись для того щоб інші користувачі могли знайти даного користувача, інше використовується для відображення в чаті. Відповідно, останнім полем, яке потрібно заповнити користувачеві, це є пароль.

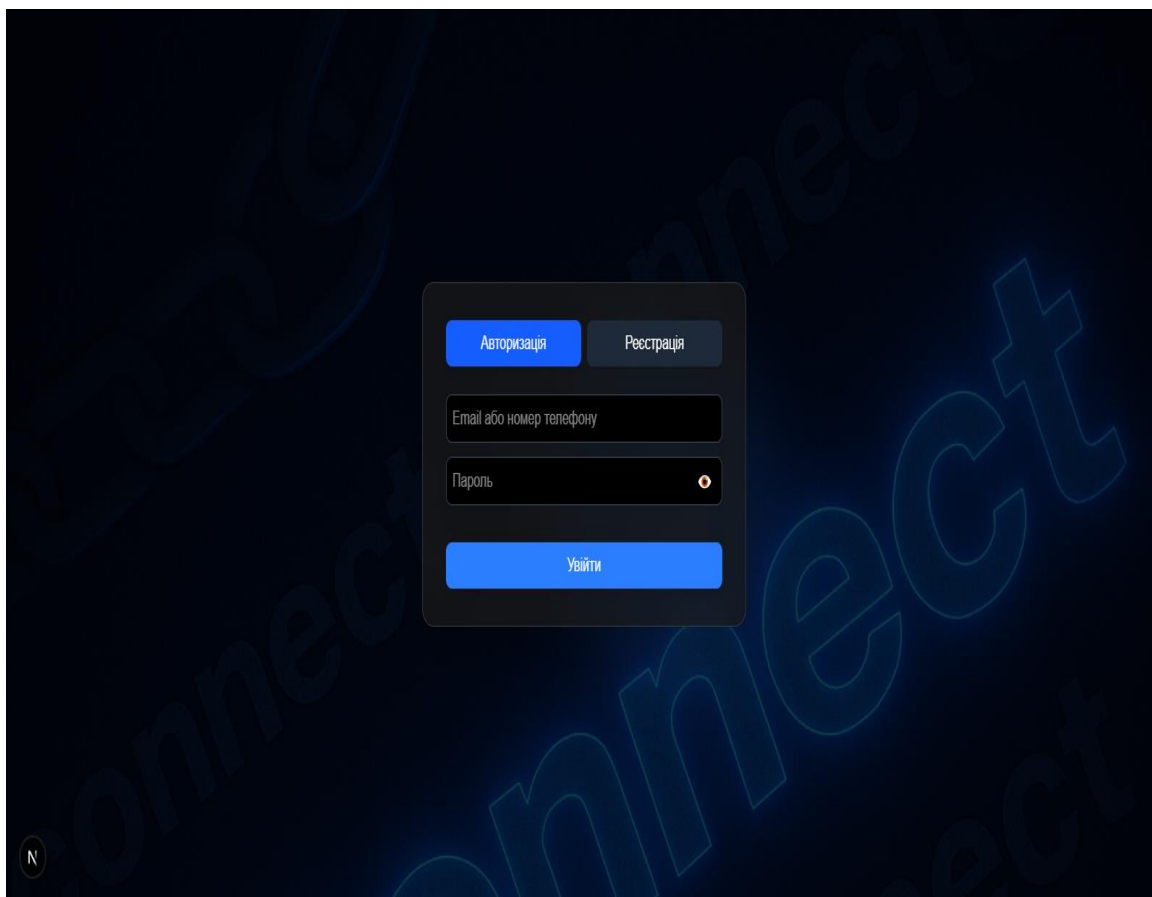


Рисунок 3.1 – Авторизація користувача

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

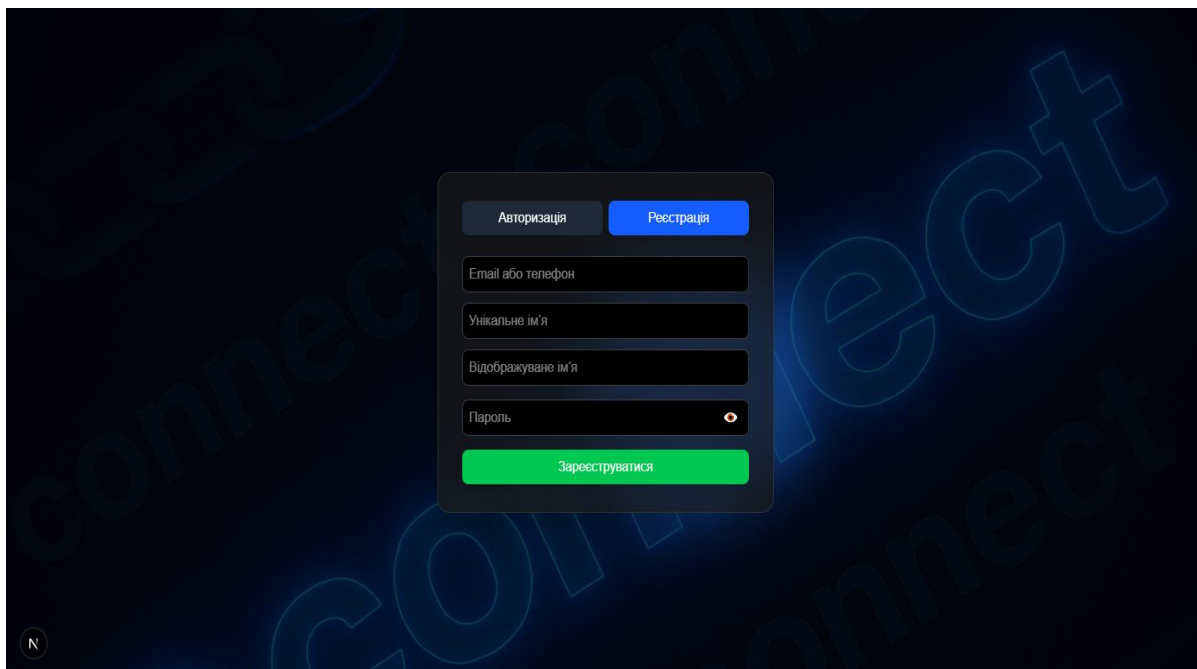


Рисунок 3.2 – Реєстрація користувача

Після того, як користувач пройде етап реєстрації, задля безпеки на його реєстраційні дані надійде сповіщення з кодом підтвердження, а саме на електронну адресу, як представлено на рис. 3.3, або на номер телефону. Відповідне зображення наведено на рис. 3.4. Система зроблена таким чином, якщо користувач не проходить етап верифікації своїх облікових даних, профіль є деактивованим. Тобто іншими словами аккаунт є створений, але задля того, щоб уникнути не санкціонованого доступу потрібно активізувати свої дані, які будуть закріплені в системі за певним користувачем. Після надходження таких даних, та заведення користувачем цих даних, він автоматично потрапляє на етап вищезазначеної авторизації, де його дані успішно активовано та він може перейти до наступного етапу.

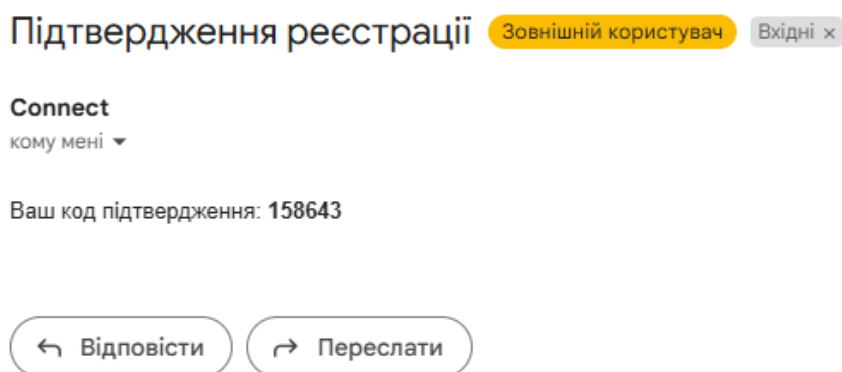


Рисунок 3.3 – Надісланий код підтвердження на пошту користувача

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

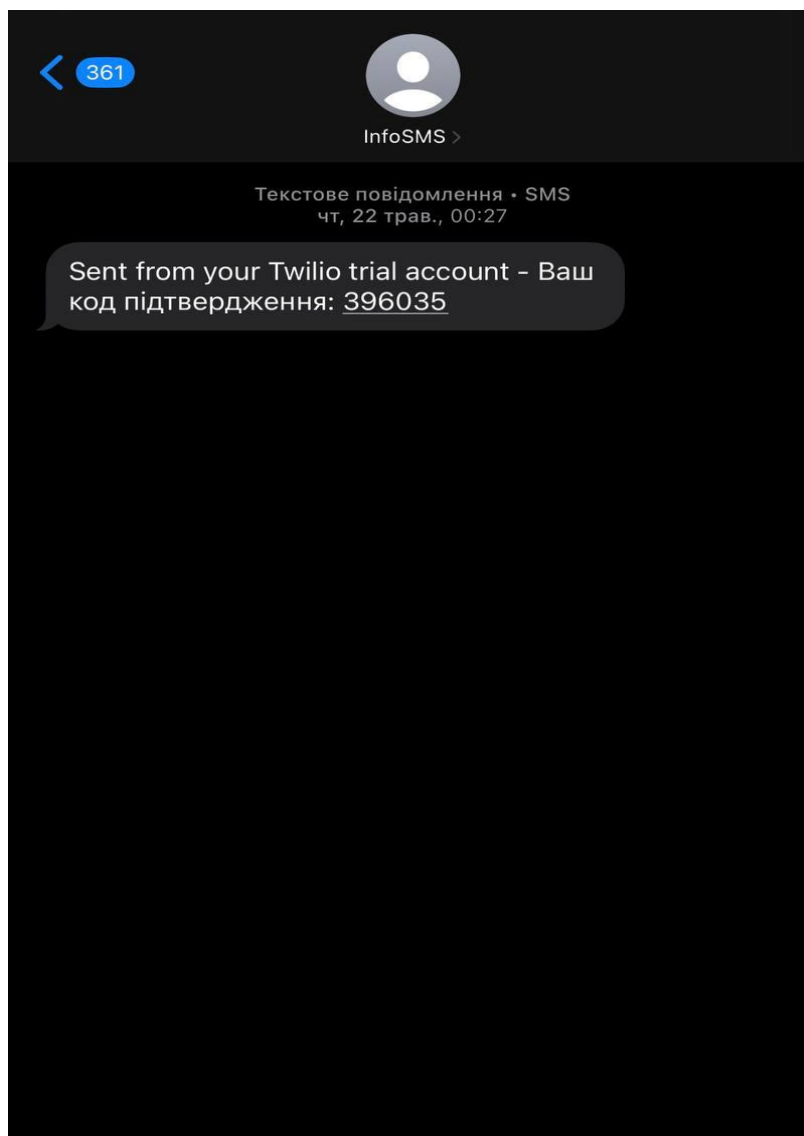


Рисунок 3.4 – Надісланий код підтвердження на номер телефону користувача

Після успішної реєстрації/авторизації користувач переходить до головної сторінки, яка зображена на рис. 3.5. На ній користувач може побачити чати з користувачами, в яких він брав участь та останні повідомлень з цих чатів. Також на цій сторінці можна побачити, чи є користувач в мережі чи він поза нею. Додатково до вищесказаного, користувач також може знайти свого співрозмовника, як показано на рис. 3.6, за його @username. При реєстрації користувачі вводять це поле та цілком логічно зрозуміло чому воно є унікальним.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

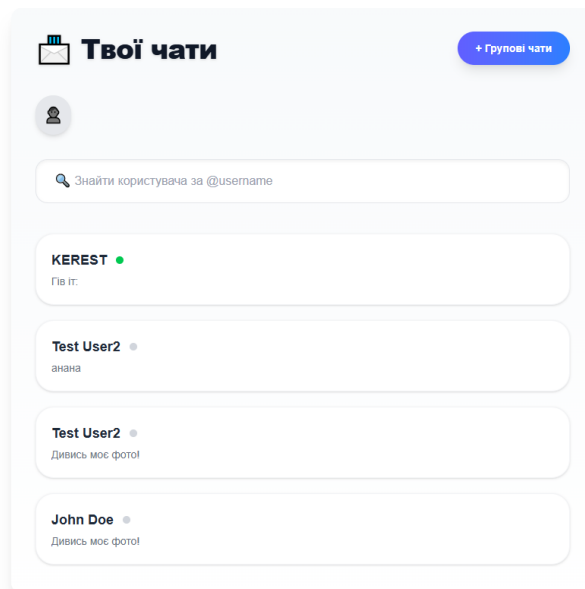


Рисунок 3.5 – Головна сторінка користувача

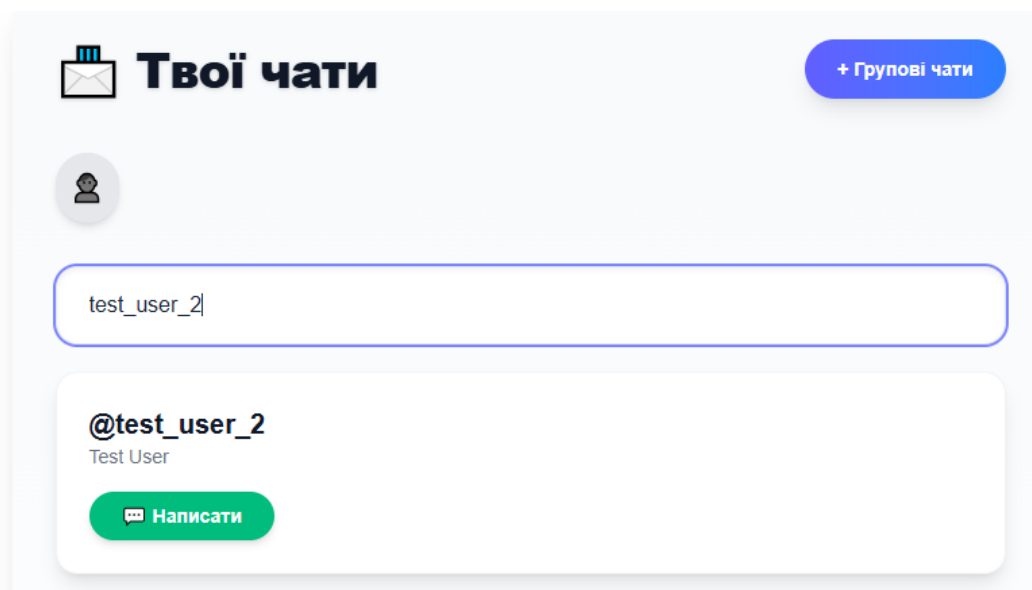


Рисунок 3.6 – Знаходження користувача за його унікальним @username

Наступний етап в системі – користувач може обрати чат з будь-яких наявних, якщо вони у нього є. В іншому випадку, він може скористатись пошуком та знайти потрібно користувача та почати з ним чат. Якщо в користувача уже є наявні чати, то він з головної сторінки обирає потрібний йому чат. Та відповідно переходить до етапу листування з користувачем. В іншому випадку, коли у користувача немає чату, він знаходить свого співрозмінка за унікальним @username та може перейти безпосередньо з пошуку у приватний чат з обраним або знайденим користувачем.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

Перейшовши відповідно до сторінки чатів, користувач може переглянути та головне розпочати листування з користувачем в режимі реального часу. Тобто, після того, як відкрилась сторінка чату, користувач може розпочати листування, як проілюстровано на рис. 3.7.

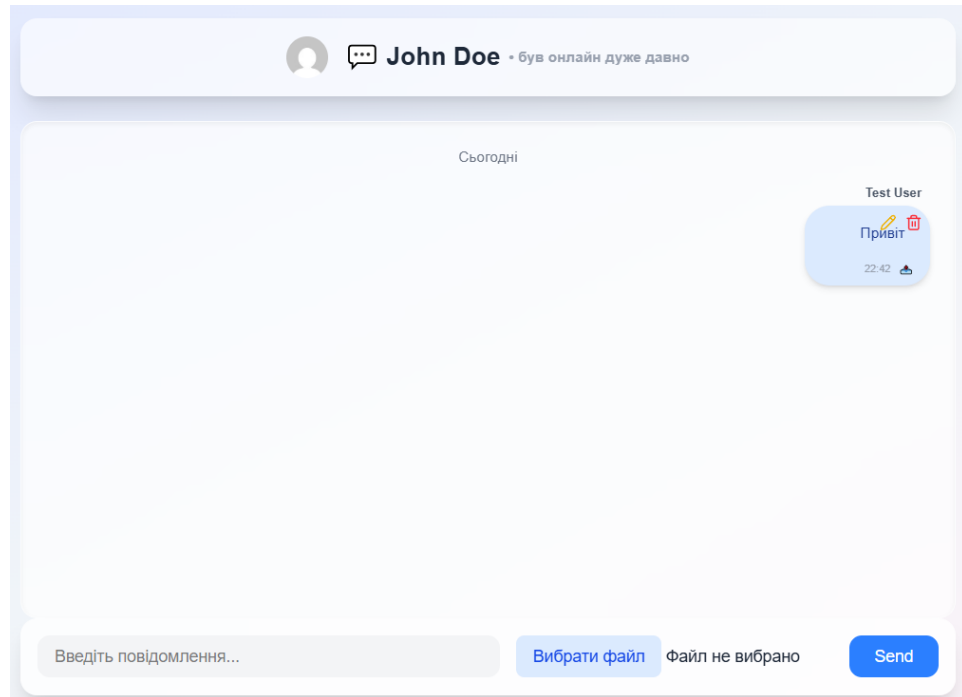


Рисунок 3.7 – Сторінка особистого чату з користувачем

Коли користувач надсилає повідомлення, він може відстежити дату його надсилання. Якщо повідомлення було надіслано сьогодні, встановлюється дата з позначкою “Сьогодні”. Проте з початком наступного дня всі повідомлення, надіслані до 00:00, тобто до початку поточного дня, набувають статусу “Вчора”. Наступні повідомлення вже не матимуть певного статусу, як-от “Сьогодні” чи “Вчора”, а замість цього буде відображатися конкретна дата листування, як показано на рис. 3.8. Також користувач може побачити статус повідомлення, якщо повідомлення успішно надіслано, з’являється іконка “”, коли ж користувач, якому було відправлено повідомлення, заходить у чат, відображається іконка “”. Це означає, що повідомлення було успішно отримано та переглянуто. Окрім цього, додатково відображаються іконки пензлика та кошика, які свідчать про можливість редагування та видалення повідомлень відповідно.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

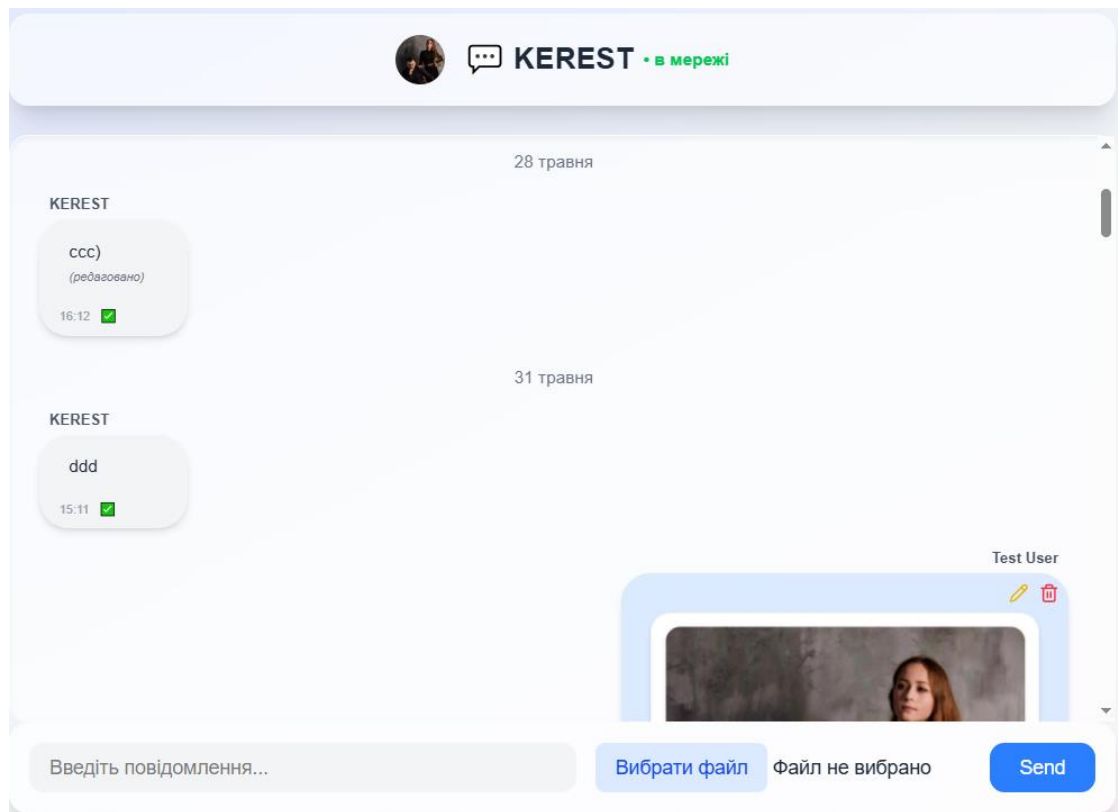


Рисунок 3.8 – Відображення дати та статусу відправленого повідомлення

Перейдемо далі. Детальніше про роботу з повідомленнями. Крім того, що їх можна переглядати та відстежувати статуси (чи повідомлення прочитано, чи надіслано), з ними також можна виконувати певні дії. Перш за все, у сучасних вебмесенджерах обов'язково повинна бути можливість редагування або видалення повідомлень. Часто трапляється, що під час спілкування користувачі не одразу помічають помилку в слові чи реченні, або ж взагалі надсилають повідомлення не тій людині. У таких випадках функція редагування повідомлень є дуже корисною. У системі також реалізована можливість редагувати повідомлення. Якщо користувач хоче змінити щось у своєму повідомленні, він може натиснути на іконку олівця, після цього повідомлення перейде в режим редагування, як зображено на рис. 3.9 та рис. 3.10.

Не менш важливою є функція видалення повідомлень. Часто буває так, що після надсилання повідомлення користувач розуміє, що воно було зайвим, або, наприклад, через пізню годину вирішує його видалити. Причин може бути безліч, тому така можливість також реалізована в системі.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				58
Змн.	Арк.	№ докум.	Підпис	Дата		

Редагування та видалення повідомлень відбувається в режимі реального часу. Не виникне ситуації, коли користувач “видав” повідомлення, але співрозмовник, увійшовши до чату пізніше, все ще бачить його. Усе працює в реальному часі: навіть якщо одержувач не перебуває онлайн і не побачив повідомлення, але відправник його видав, тому воно автоматично зникає і в отримувача.

Приклад видалення повідомлення можна побачити на рис. 3.11 та рис. 3.12.

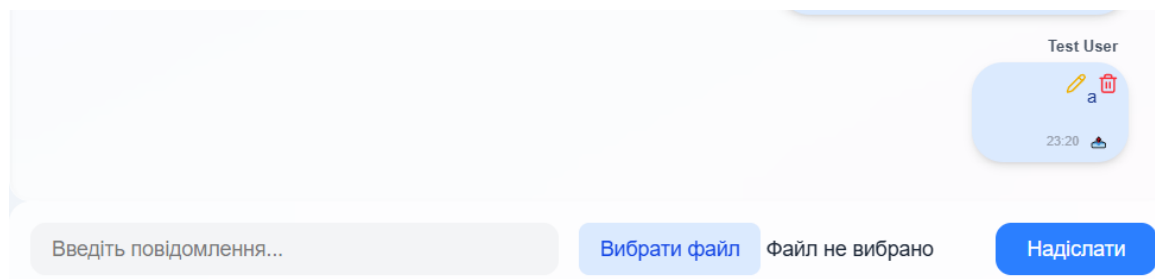


Рисунок 3.9 – Повідомлення до оновлення

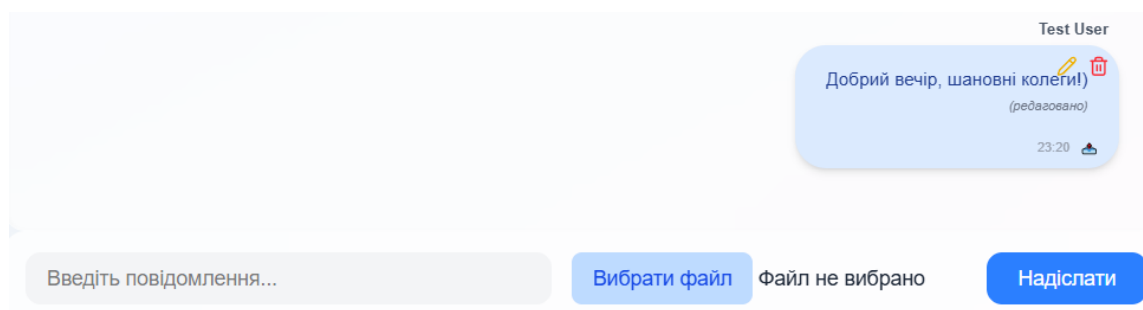


Рисунок 3.10 – Повідомлення після оновлення з встановленням статусу “редаговано”

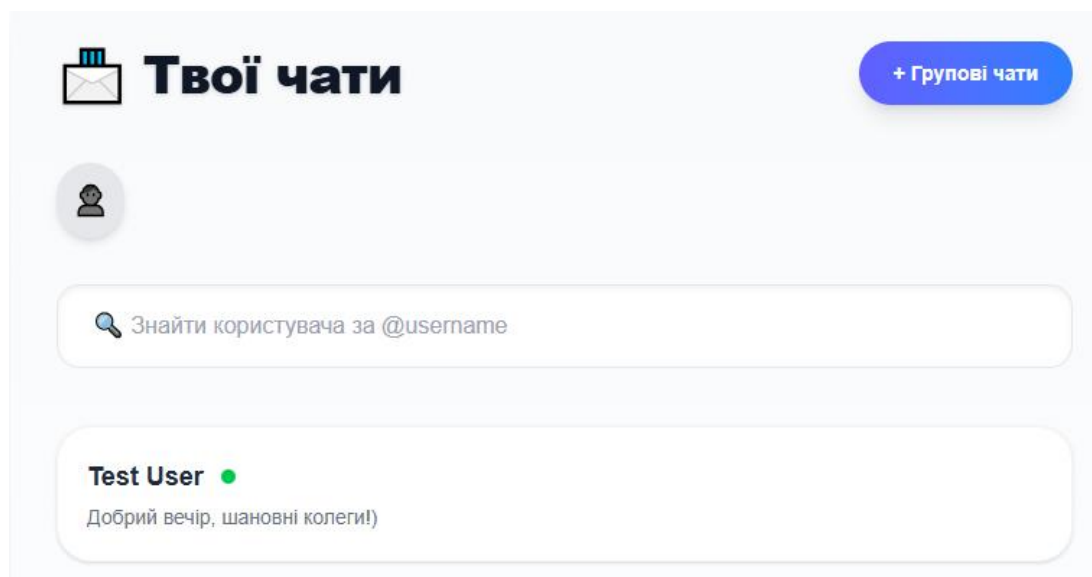


Рисунок 3.11 – Повідомлення до видалення

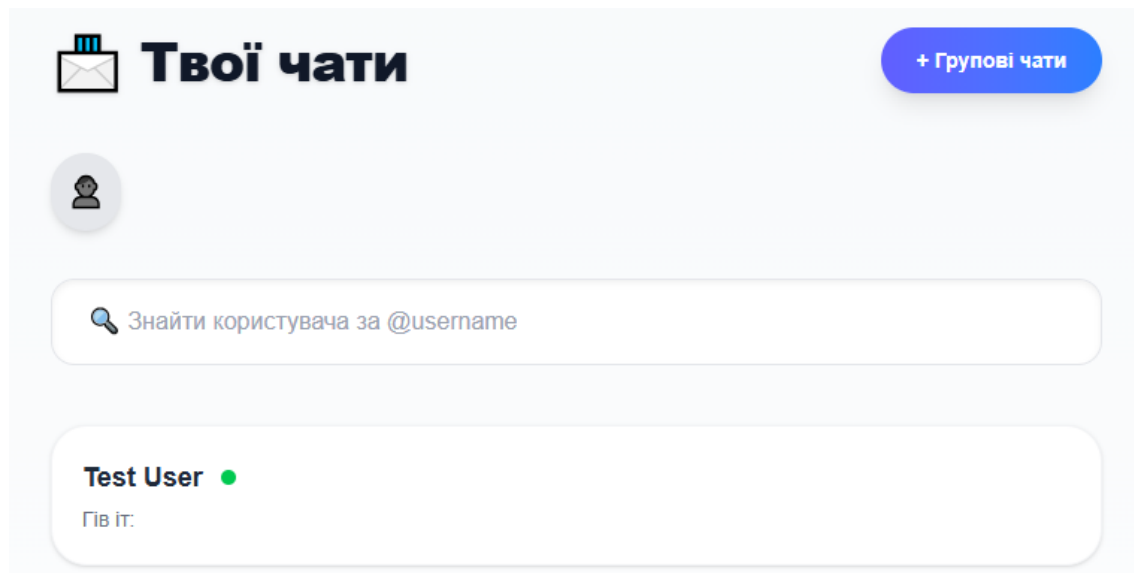


Рисунок 3.12 – Повідомлення, яке було видалено. Відображається повідомлення, яке було до видаленого

Перейдемо до наступного етапу розробки вебмесенджера. Більшість людей, які користуються месенджерами, принаймі одного разу надсилали у чатах певний графічний контент співрозмовнику. Звісно, це дуже важлива функція у сучасному технологічному світі, яка надає можливість надіслати фотографію, важливий документ або відео. Було вирішено не відставати від сучасних тенденцій у розвитку месенджерів, тому потрібно врахувати необхідність реалізації функції передавання файлів між користувачами.

Перш за все, для коректної передачі файлів важливо, щоб користувач мав змогу попередньо переглянути вміст файла й переконатися у його допустимості. Тому, наприклад, файли у форматі PDF або текстові документи, як зображено на рис. 3.13, можна буде переглядати до їх завантаження.

Фотографії, як показано на рис. 3.14, доступні для перегляду безпосередньо зі сторінки чату. Користувач може просто відкрити відео (див. рис. 3.15), чи зображення і переглянути його вміст. Далі рішення за користувачем – або завантажувати контент або ні.

Також реалізовано можливість завантаження інших типів файлів, а саме архіви, документи у форматах *.docx, *.xlsx, *.pptx та інші.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				60
Змн.	Арк.	№ докум.	Підпис	Дата		

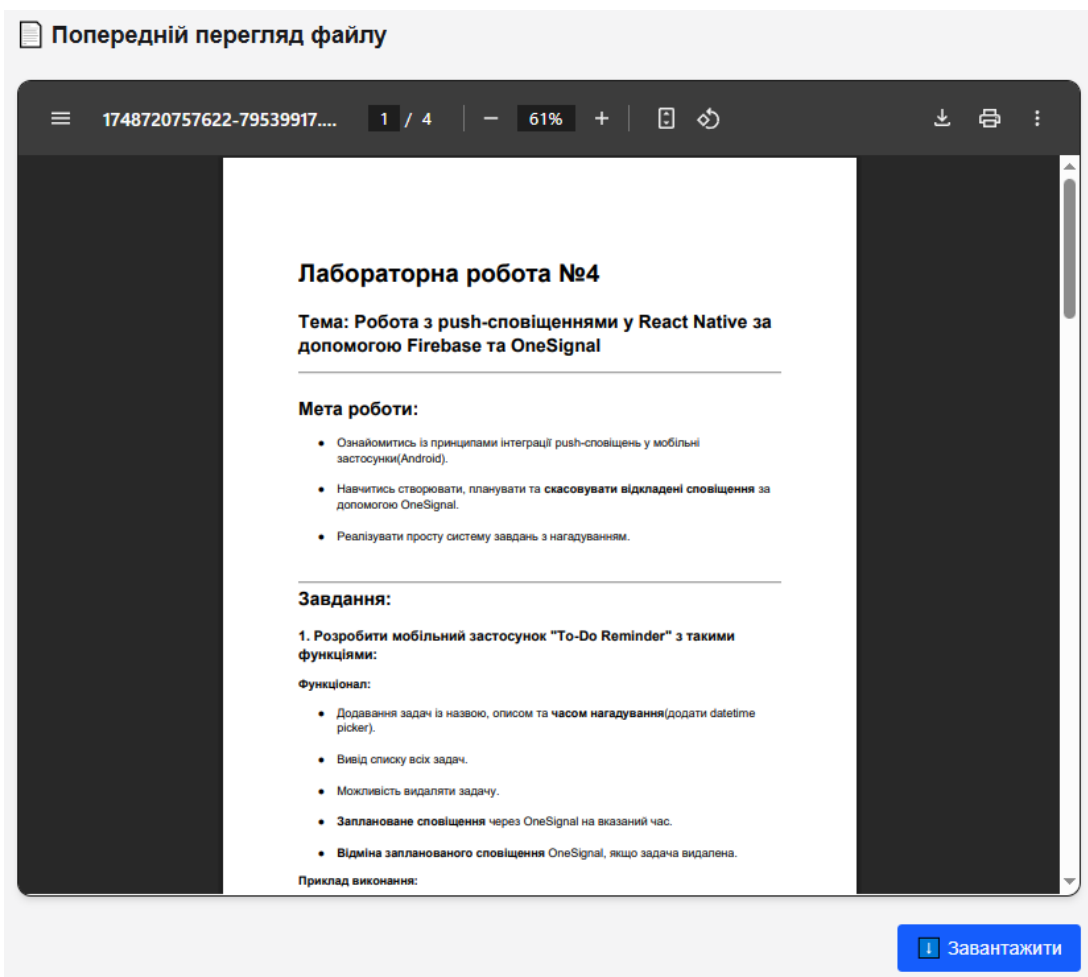


Рисунок 3.13 – Попередній перегляд файлу *.pdf



Рисунок 3.14 – Попередній перегляд фото

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

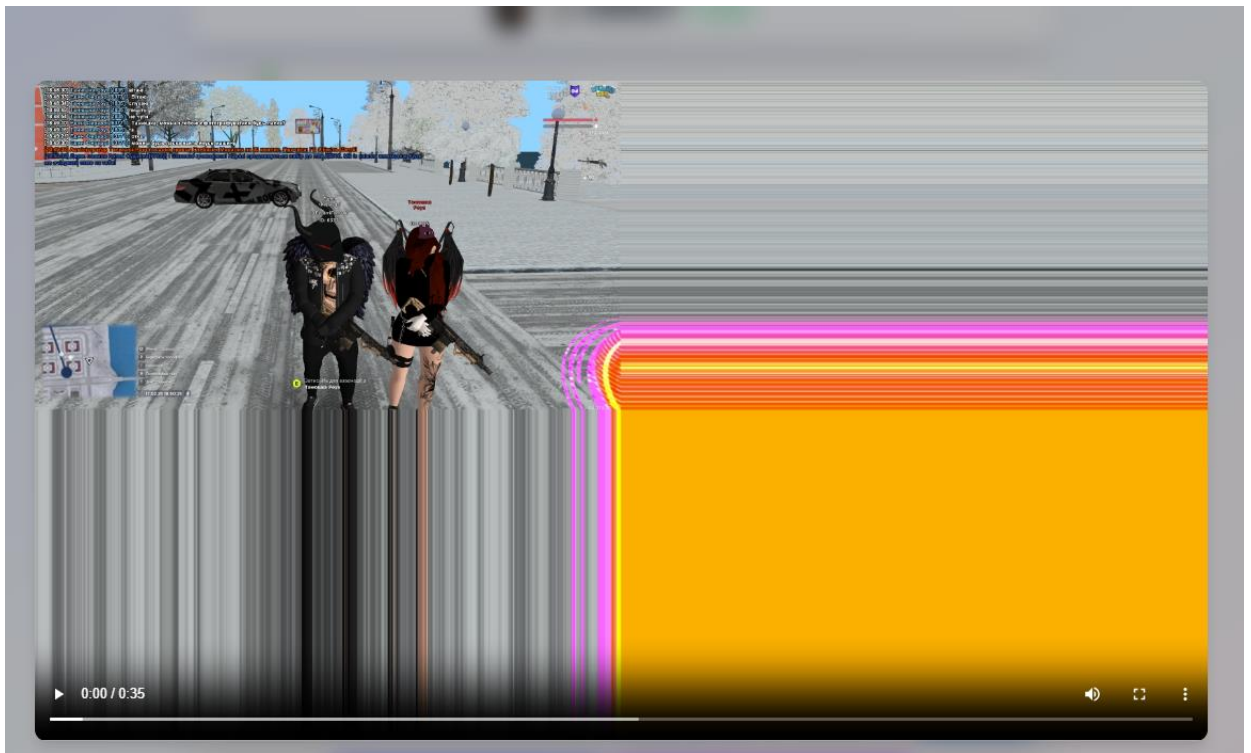


Рисунок 3.15 – Попередній перегляд відео

У вебмесенджері також доступна можливість прослуховування аудіофайлів у форматі *.mp3 безпосередньо зі сторінки чату, а також попередній перегляд файлів у форматах *.txt, *.md тощо. Для інших типів файлів реалізоване пряме завантаження зі сторінки чату, як показано на рис. 3.16.

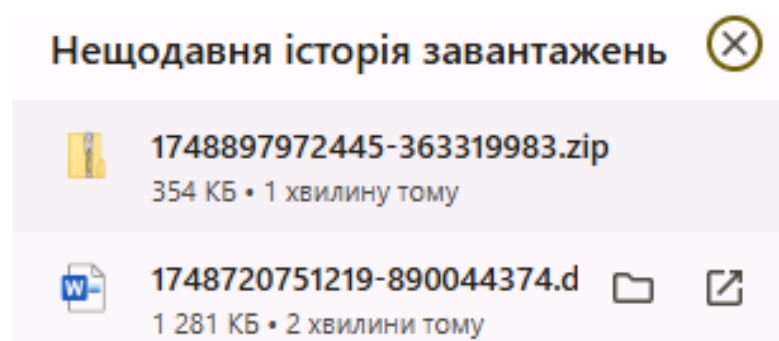


Рисунок 3.16 – Завантаження файлів

У вебмесенджері також реалізовано функцію перегляду останнього входу користувача та кількості часу, протягом якого він був неактивним, що представлено на рис. 3.17. При натисканні на користувача відкривається його профіль, що проілюстровано на рис. 3.18, у якому відображаються базові

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

налаштування: чи пройшов користувач верифікацію, коли був здійснений останній вхід у систему та дата реєстрації користувача в системі.



Рисунок 3.17 – Кількість неактивного часу користувача у системі

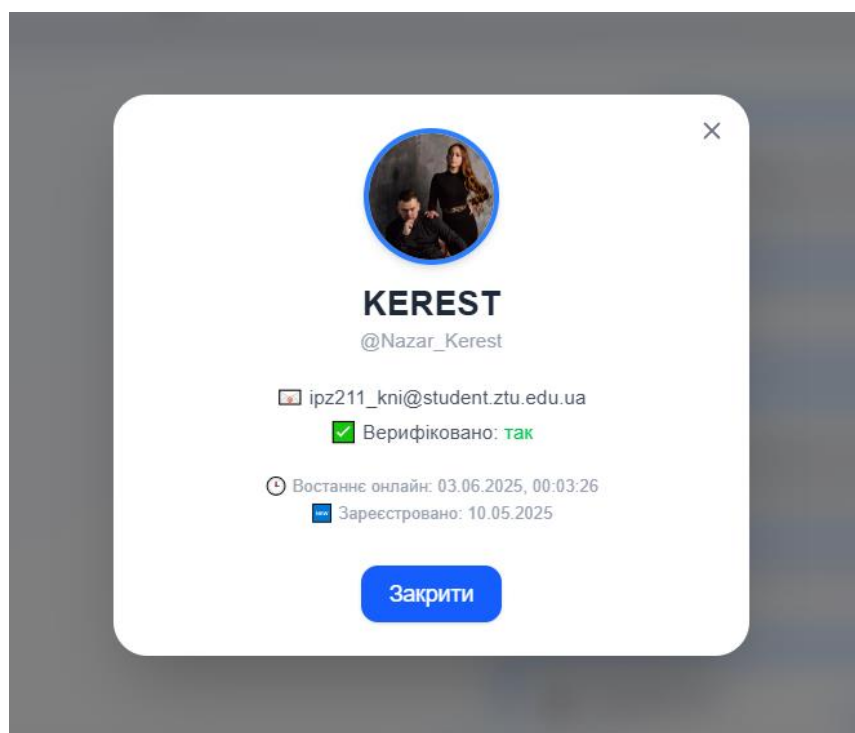


Рисунок 3.18 – Профіль користувача та базові дані про нього

Перейдемо до наступної частини функціоналу вебмесенджера. У сучасному світі месенджерів групові чати вже не є новинкою. Зазвичай, коли користувачі спілкуються в особистих чатах, їхня комунікація відбувається приватно, тобто без зайвих очей. Було вирішено додати можливість спілкування в групах, адже надсилати одну й ту ж саму інформацію кільком користувачам окремо нелогічно. Саме для цього й існують групові чати. Тепер, коли користувач заходить на сторінку групових чатів, як показано на рис. 3.19, він може побачити, в яких групах є учасником. Також на цій сторінці реалізована можливість створення нової групи з будь-якими учасниками, як зображено на рис. 3.20. Водночас, можна спочатку створити групу без учасників, а вже потім поступово додавати їх.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				63
Змн.	Арк.	№ докум.	Підпис	Дата		

 **Групові чати**

+

Створити групу

TEST

Nazar_Kerest: ++



Назарчик чіНАзес 100

test_user_2: Test



← Повернутись до сторінки чату

N

Рисунок 3.19 – Сторінка групових чатів користувача

 **Групові чати**

+

Створити групу

Назарчик чіНАзес 100

test_user_2: Test



TEST

Nazar_Kerest: ++



← Повернутись до сторінки чату

Рисунок 3.20 – Створення групового чату з переліком учасників

Після того, як користувач натисне кнопку створення групи, вона автоматично відображатиметься в реальному часі в усіх доданих учасників. Ілюстрацію цієї функції наведено на рис. 3.21.

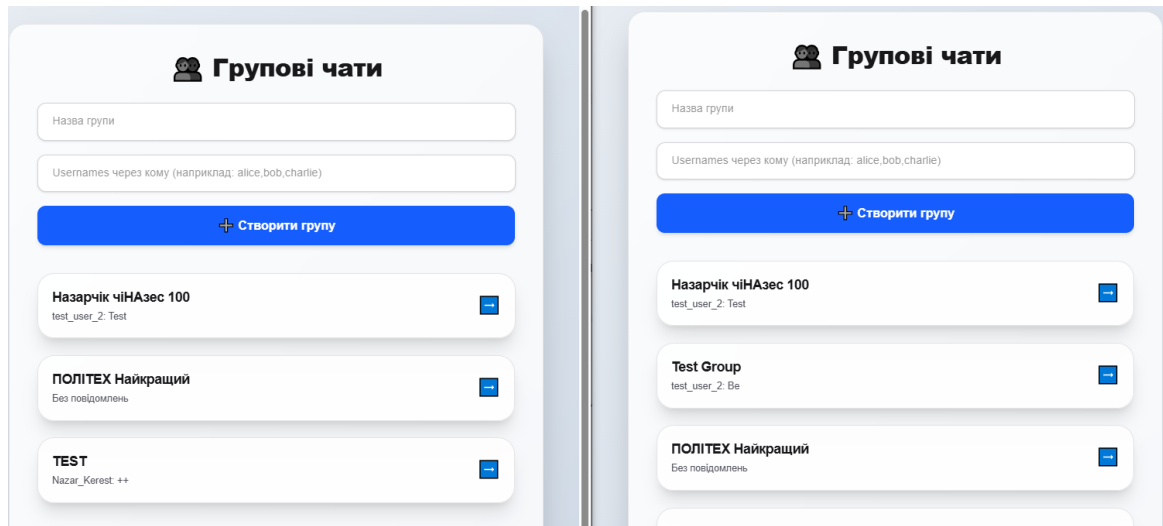


Рисунок 3.21 – Відображення у двох різних користувачів створенної групи

Після створення групи користувач може увійти до неї, як показано на рис. 3.22, та розпочати спілкування або перейти до адміністрування групи, зокрема, до управління її учасниками.

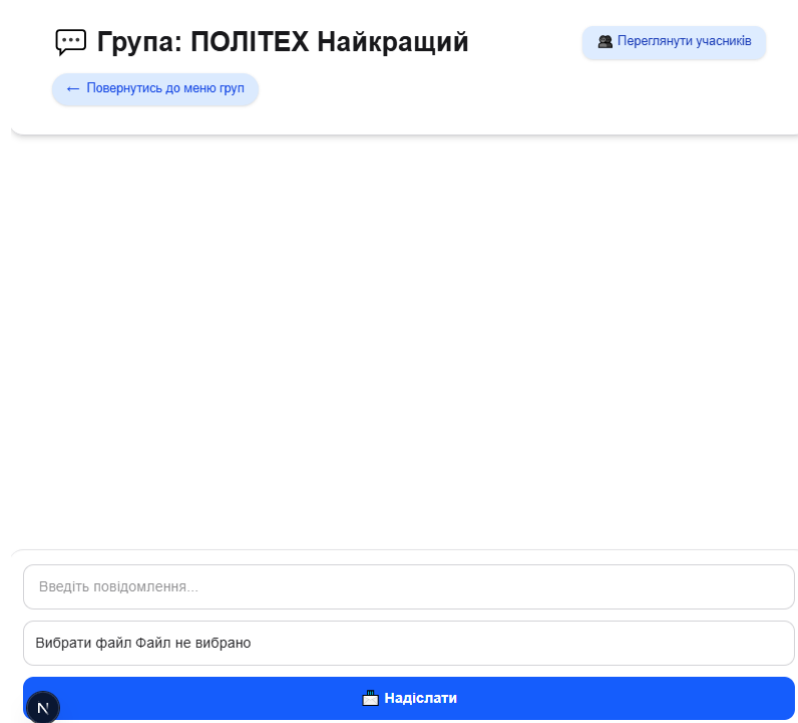


Рисунок 3.22 – Новостворена група

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				65
Змн.	Арк.	№ докум.	Підпис	Дата		

Далі розпочинається безпосередня комунікація між учасниками групи, ця можливість показана на рис. 3.23. У групі також доступний обмін різним контентом, а також функція відстеження часу доставки та дати надсилання повідомлень.

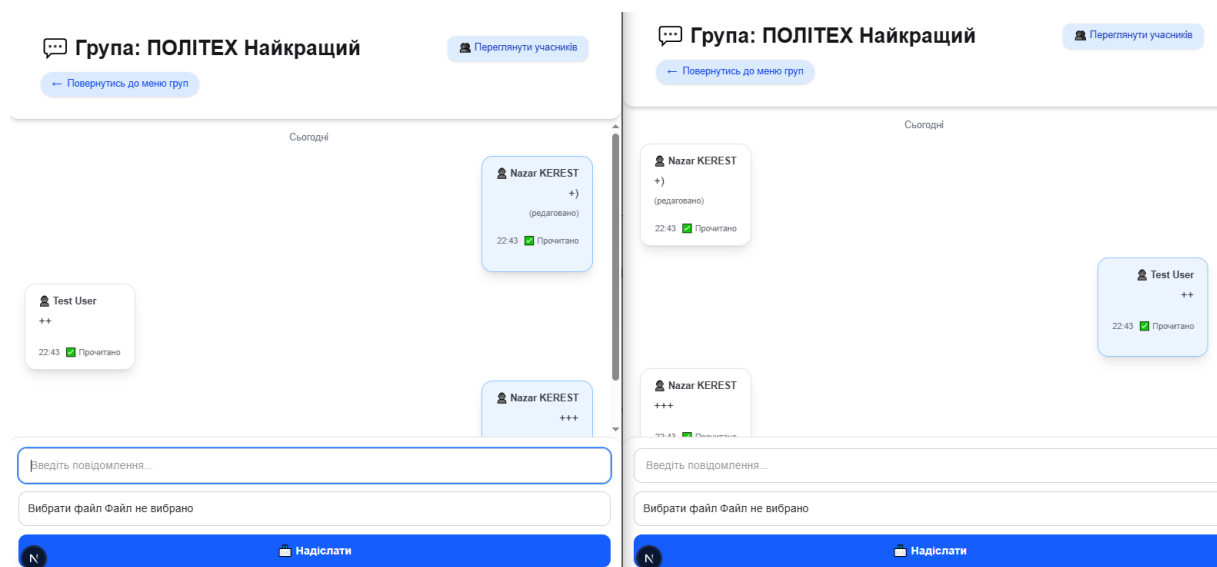


Рисунок 3.23 – Комунікація між користувачами, можливість відстеження дати, часу, доставки повідомлення

У групових чатах також доступна можливість редагування та видалення повідомлень. Додатково реалізовано функції адміністрування групи. Зазвичай для уникнення неприємних ситуацій у групах передбачено адміністрування. Користувачі можуть надсилати повідомлення, ділитися контентом, однак адміністратори мають розширені повноваження. Вони можуть призначати нових адміністраторів або видаляти їх. Коли адміністратор призначає іншого користувача адміністратором, у цього користувача з'являється доступ до функцій керування групою зокрема, зміни назви, управління учасниками тощо. З міркувань безпеки адміністратор не має права видаляти інших адміністраторів, лише звичайних учасників. Повний контроль над групою має лише її власник. Саме він може призначати або знімати адміністраторів, видаляти користувачів із групи тощо. Власник та інші учасники також можуть переглядати список учасників групи, разом із кількістю, як показано на рис. 3.24.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				66
Змн.	Арк.	№ докум.	Підпис	Дата		

Власник групи має змогу призначити будь-якого учасника адміністратором. Після цього у призначеного адміністратора одразу відкривається доступ до функцій керування групою, що представлено на рис. 3.25, включно з можливістю додавання або видалення користувачів. Якщо ж буде призначено ще одного адміністратора, то він, у свою чергу, не зможе зняти з посади іншого адміністратора або видалити його, як це продемонстровано на рис. 3.26. Крім того, адміністратору після призначення надається можливість змінювати назву групи. Після оновлення назва автоматично змінюється в усіх учасників чату, що проілюстровано на рис. 3.27.

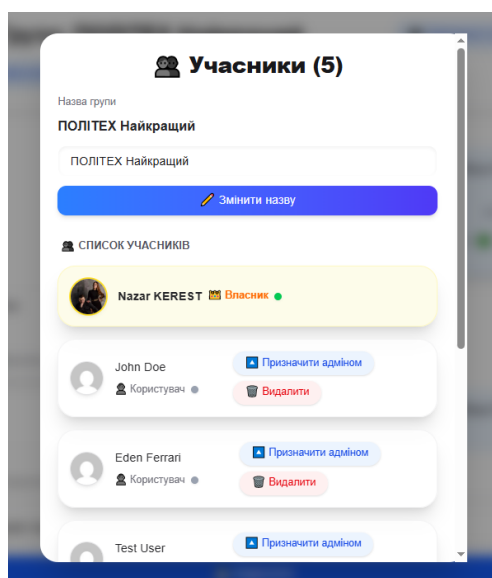


Рисунок 3.24 – Перегляд учасників групового чату

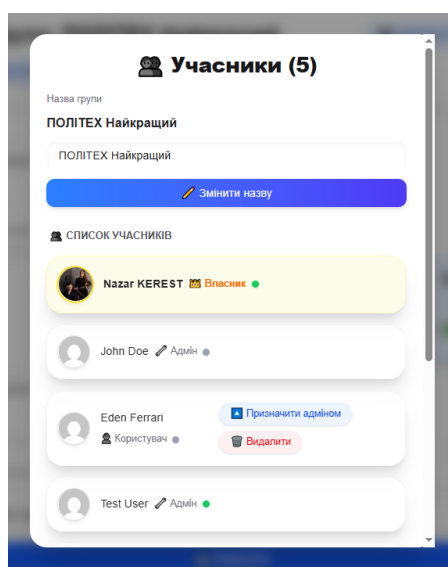


Рисунок 3.25 – Керування групою на посаді адміністратора

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				67
Змн.	Арк.	№ докум.	Підпис	Дата		

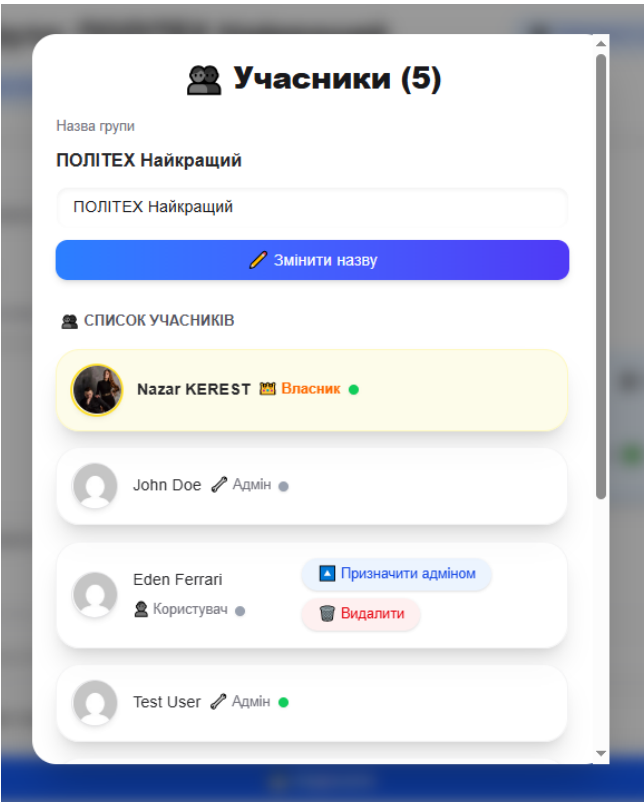


Рисунок 3.26 – Якщо назначено нового адміністратора, тоді інший адміністратор ніяк не зможе з ним взаємодіяти

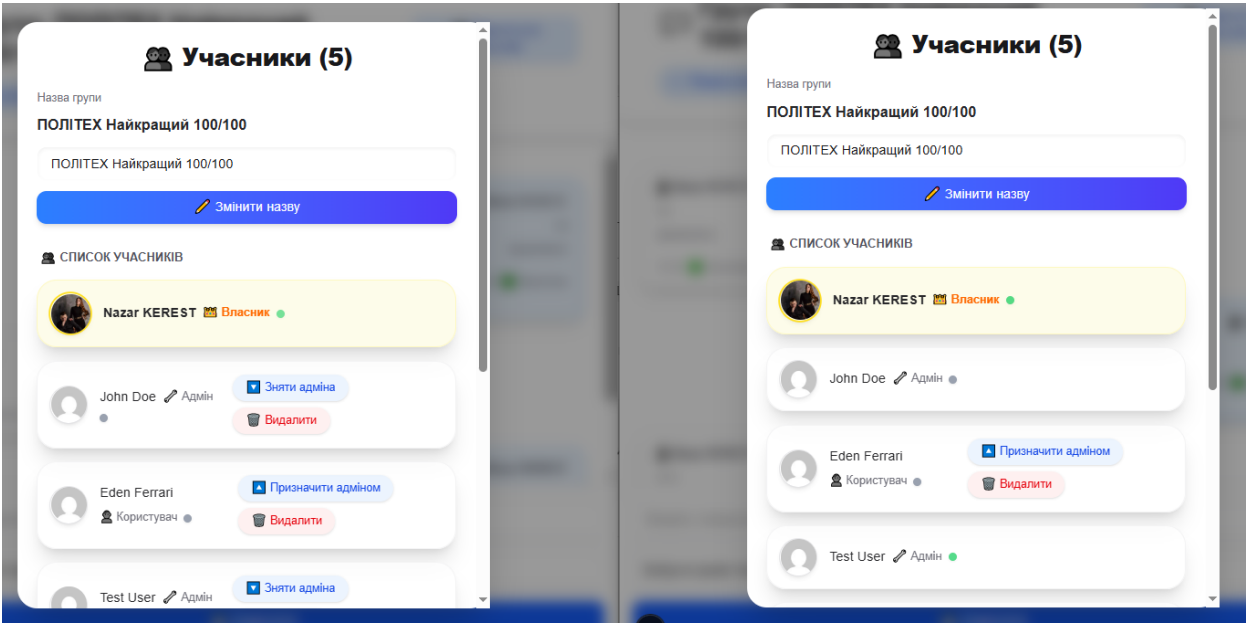


Рисунок 3.27 – Автоматичне оновлення назви групи

У груповому чаті також реалізована можливість самостійного виходу користувача з групи, як видно на рис. 3.28, а також можливість видалення всієї групи, що можна побачити на рис. 3.29. Варто зауважити, що функція повного

видалення групи доступна виключно власнику, що зроблено з міркувань безпеки, ніхто інший скористатися цією опцією не зможе. Під час видалення користувача з групи, якщо він перебуває на сторінці групового чату, йому одразу надходить сповіщення про видалення, як показано на рис. 3.30. Якщо ж користувач перебуває на іншій сторінці, група автоматично зникає з його інтерфейсу без додаткового повідомлення. Щодо повного видалення групи, всім учасникам, які на момент видалення перебувають у групі, надходить сповіщення про її ліквідацію. Це показано на рис. 3.31. Звичайно, при створенні нової "порожньої" групи, тобто без учасників, необхідною є можливість додавати користувачів. Як видно з рис. 3.32, у меню керування групою передбачено функцію додавання учасників за їхнім "username". Це дозволяє швидко й зручно наповнювати групу потрібними учасниками.

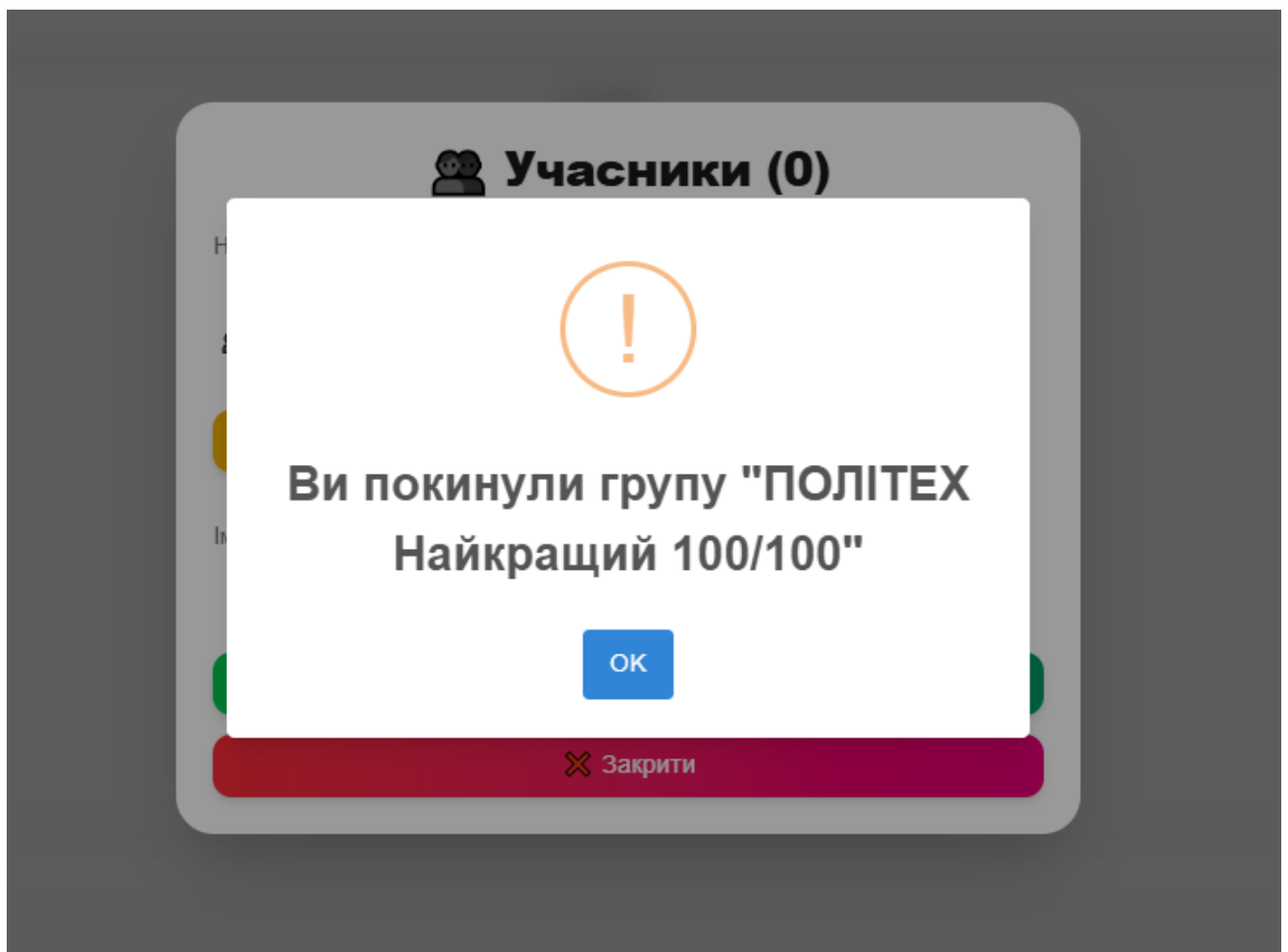


Рисунок 3.28 – Вихід з групи користувачем

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				69
Змн.	Арк.	№ докум.	Підпис	Дата		

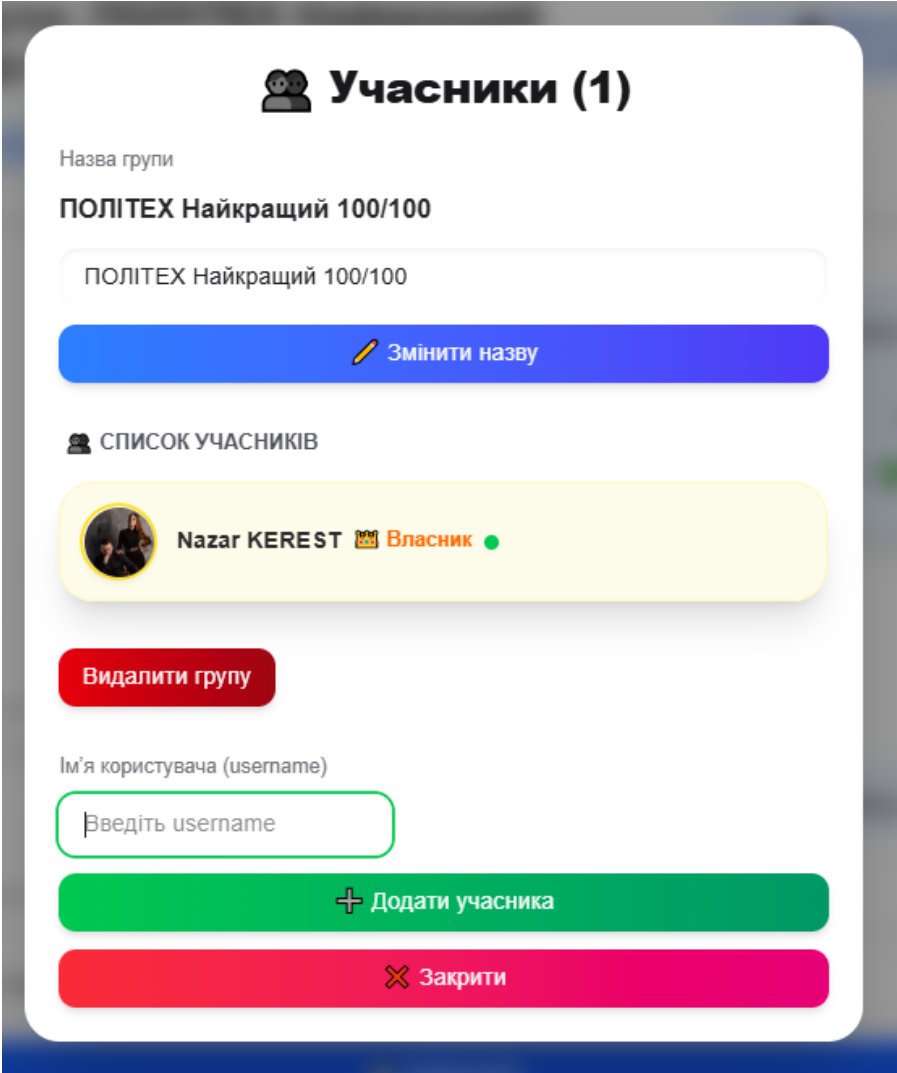


Рисунок 3.29 – Видалення групи власником групи

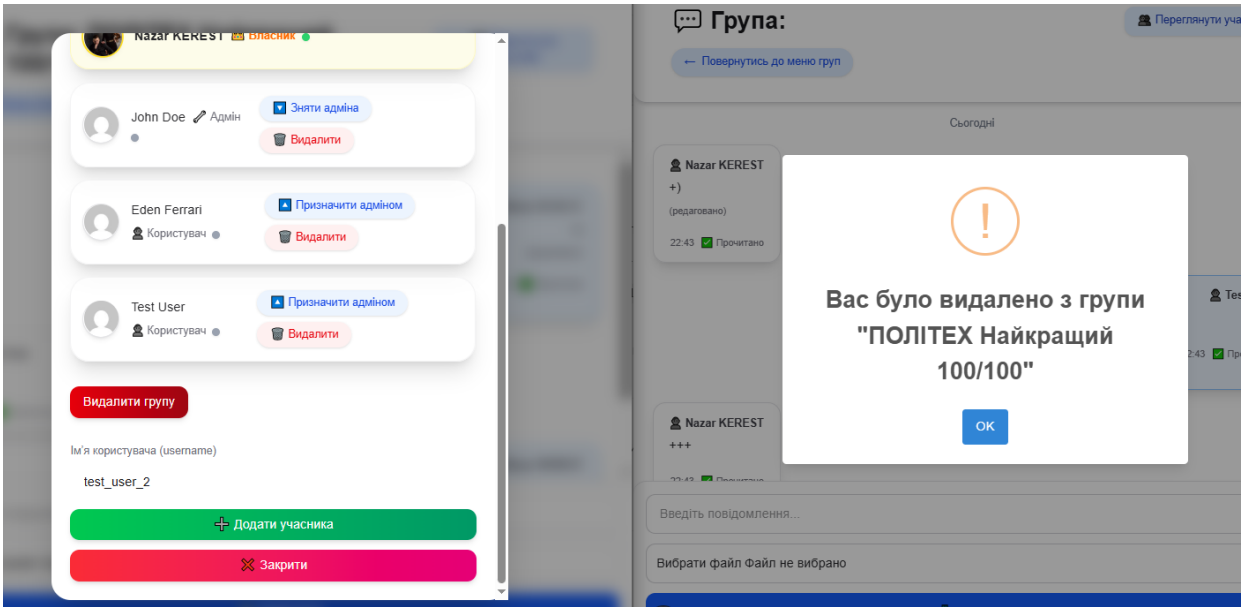


Рисунок 3.30 – Видалення користувача з групи власником групи

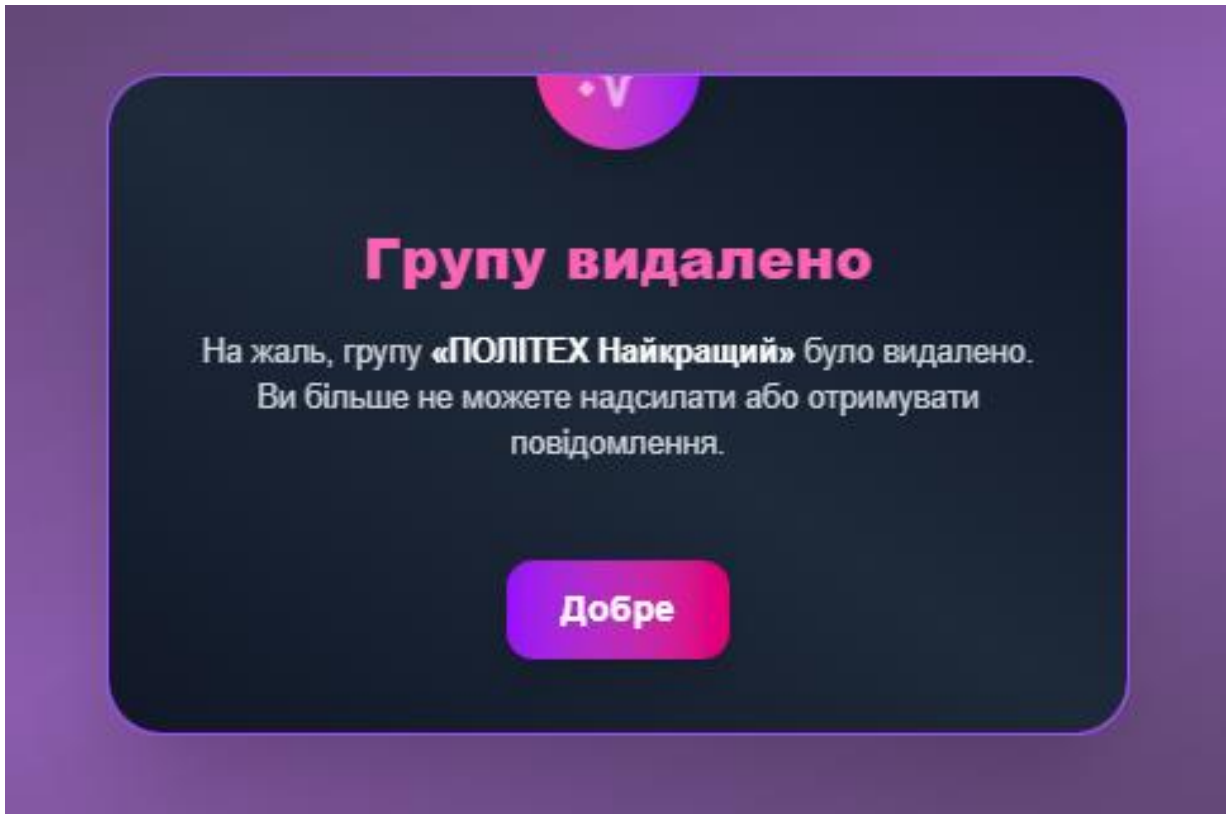


Рисунок 3.31 – Сповіщення про видалення групи

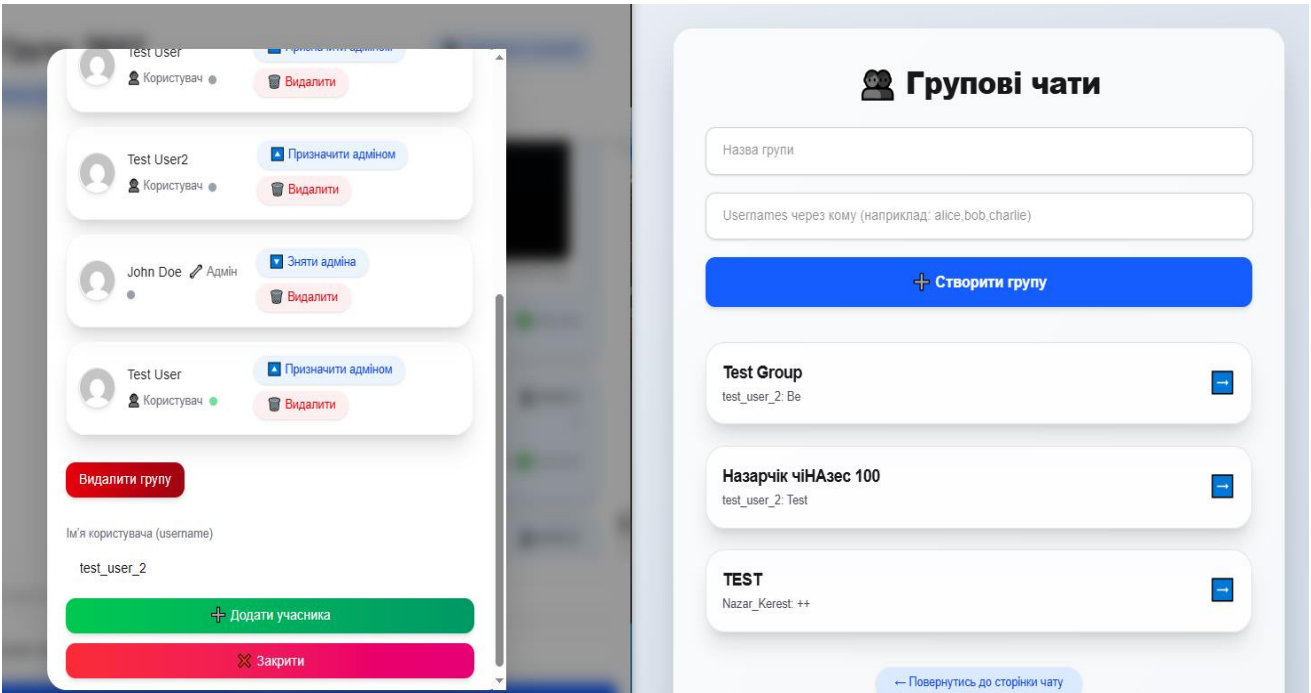


Рисунок 3.32 – Додавання користувача до групи за допомогою групового меню та автоматичне оновлення груп у доданого користувача

У груповому меню також є можливість переглянути профіль користувача, що видно на рис. 3.33, а також переглянути його статус, як зображено на рис. 3.34. Тобто користувач може бути або в мережі, або офлайн.

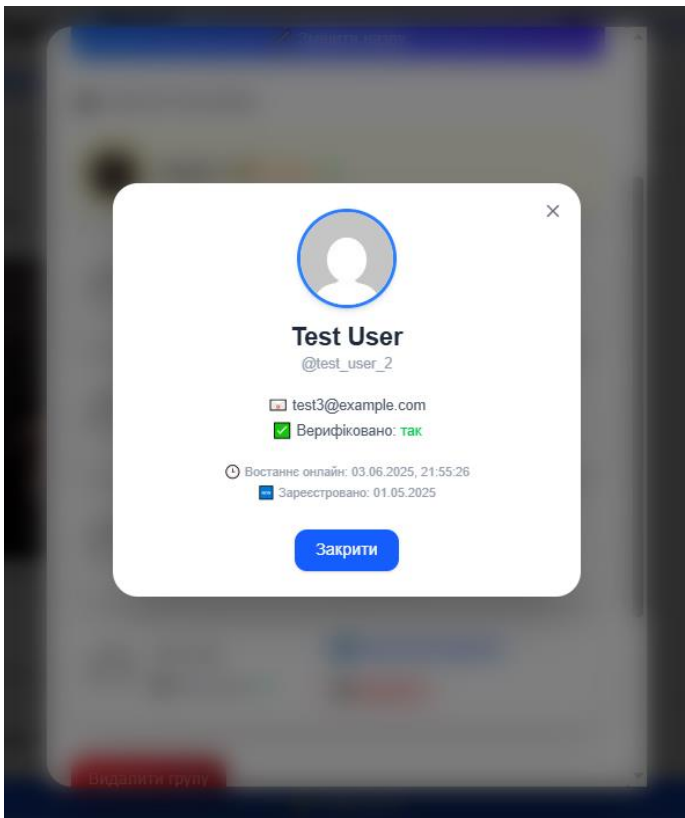


Рисунок 3.33 – Відображення профілю користувача в груповому чаті

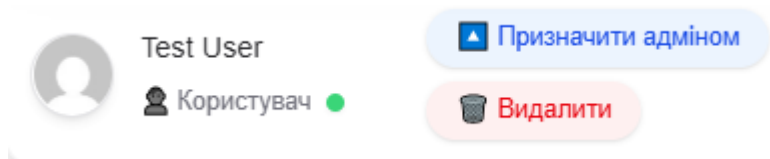


Рисунок 3.34 – Відображення статусу користувача: в мережі або ні

У системі вебмесенджера передбачена окрема сторінка профілю, як продемонстровано на рис. 3.35, де користувач може переглянути свої дані або змінити їх. Для завантаження аватарів використовується сервіс Cloudinary. Зміни відбуваються в режимі реального часу, тобто, іншими словами, коли користувач оновлює свої профільні дані, вони одразу оновлюються і підтягуються для інших користувачів, як показано на рис. 3.36.



Ім'я: KEREST

Username: @Nazar_Kerest

Зареєстрований: 10.05.2025

✎ Редагування профілю

Ім'я

KEREST

Username

Nazar_Kerest

Аватар

📎 Завантажити

PNG, JPG, JPEG

💾 Зберегти

← Повернутись до чату

Рисунок 3.35 – Сторінка профілю користувача



Ім'я: Nazar KEREST

Username: @Nazar_Kerest

Зареєстрований: 10.05.2025

✎ Редагування профілю

Ім'я

Nazar KEREST

Username

Nazar_Kerest

Аватар

📎 Завантажити

PNG, JPG, JPEG

💾 Зберегти

✅ Профіль оновлено успішно!



Nazar KEREST

@Nazar_Kerest

📧 ipz211_kni@student.ztu.edu.ua

✅ Верифіковано: так

🕒 Востаннє онлайн: 03.06.2025, 22:04:49

📅 Зареєстровано: 10.05.2025

Закрити

Рисунок 3.36 – Відображення змін після зміни даних користувацького профілю

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				73
Змн.	Арк.	№ докум.	Підпис	Дата		

Додатково у вебмесенджері є можливість спілкуватися зі штучним інтелектом, як показано на рис. 3.37.

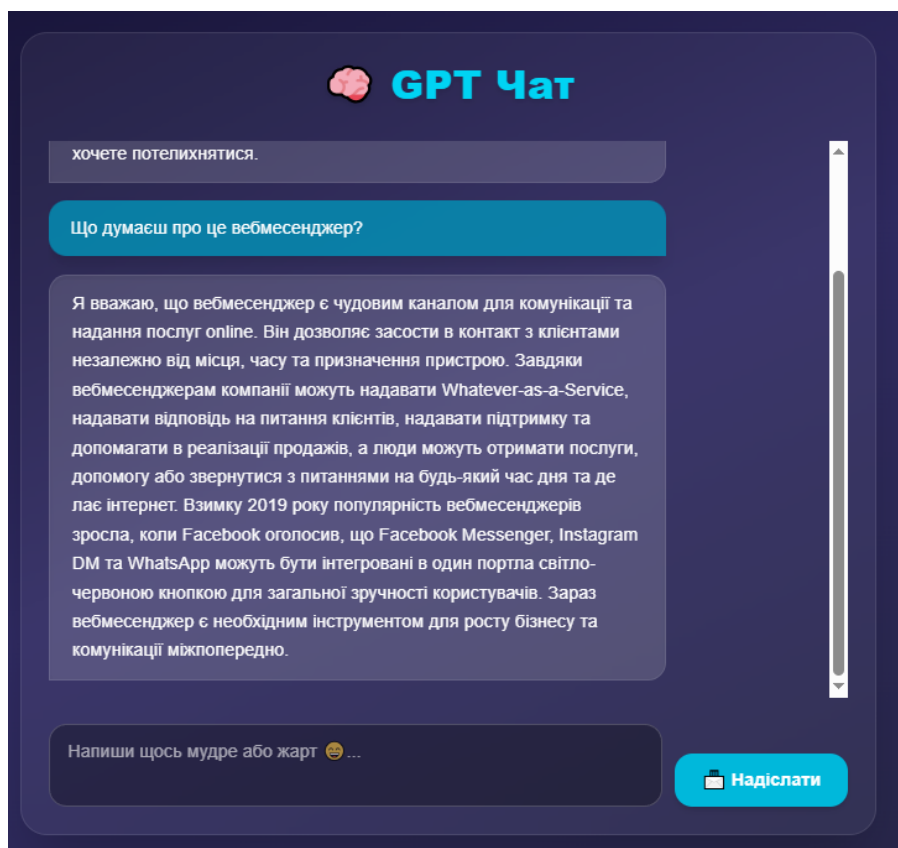


Рисунок 3.37 – Чат з штучним інтелектом

Також у системі передбачена можливість спільного перегляду відео між користувачами, що ілюстровано на рис. 3.38.

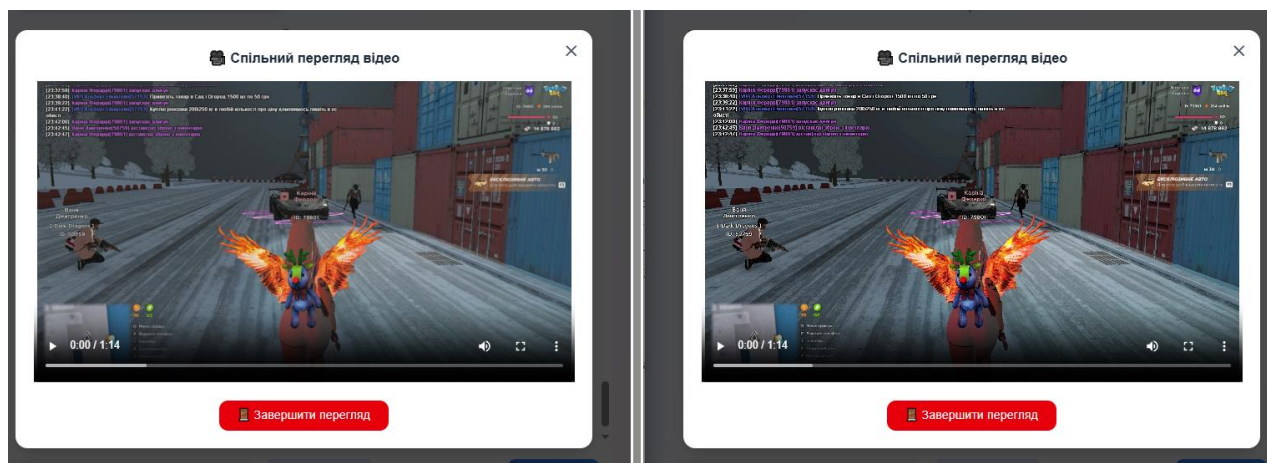


Рисунок 3.38 – Спільний перегляд відео з користувачем.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

3.2 Тестування системи

Важливо зазначити, що для таких проєктів важливою складовою є тестування. У такому проєкті потрібно перевіряти всі можливі ризики, які може спричинити користувач, та успішно їх обробляти. Перш за все, при реєстрації користувача необхідно перевірити, чи введені ним дані дійсно будуть доступні для системи. Тобто, перевірити унікальність та правильність введення номера телефону чи адреси електронної пошти, як показано на рис. 3.39. Також потрібно перевірити унікальне ім'я користувача, що відображено на рис. 3.40, за яким інші користувачі в подальшому зможуть здійснювати пошук. І, звичайно, потрібно переконатися, що пароль є надійним і важкодоступним для злоумисників, як можна побачити на рис. 3.41.

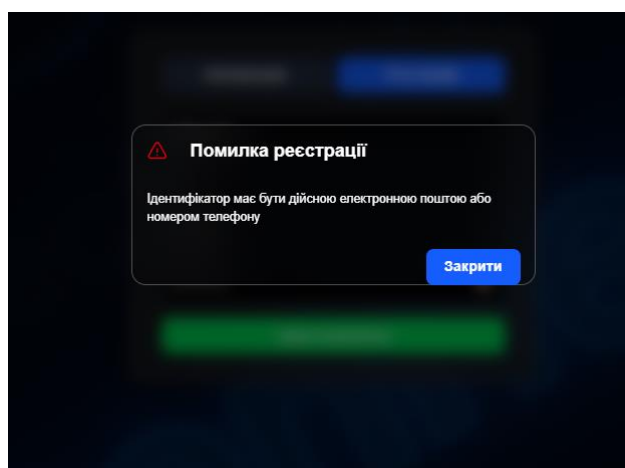


Рисунок 3.39 – Неправильність введення електронної пошти або номеру телефону

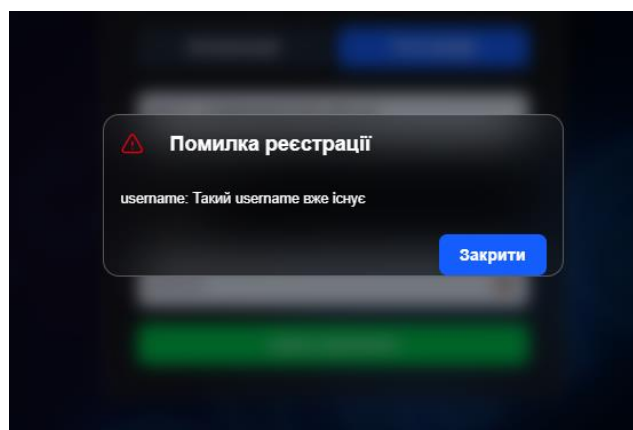


Рисунок 3.40 – Перевірка унікального імені користувача

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				75
Змн.	Арк.	№ докум.	Підпис	Дата		

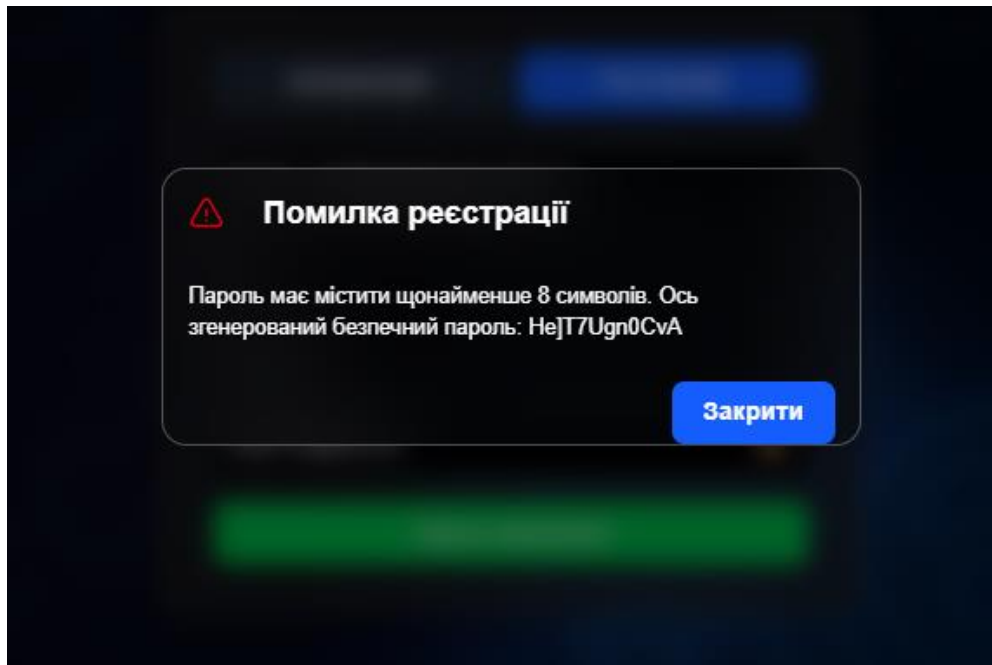


Рисунок 3.41 – Перевірка валідності та складності паролю. Автоматичний підбір та підставлення рекомендованого паролю

Також важливо обробляти помилки на початку логіну користувача, якщо введені дані не відповідають його акаунту. Наприклад, якщо вводяться дані, яких немає в системі, буде виведено повідомлення про те, що такого користувача не знайдено, як це показано на рис. 3.42. Якщо ж акаунт існує, система перевірятиме правильність введених даних при реєстрації та їх відображення під час авторизації. Іншими словами, якщо користувач введе неправильний пароль, йому повернеться повідомлення про помилку неправильного паролю, відповідно до рис. 3.43.

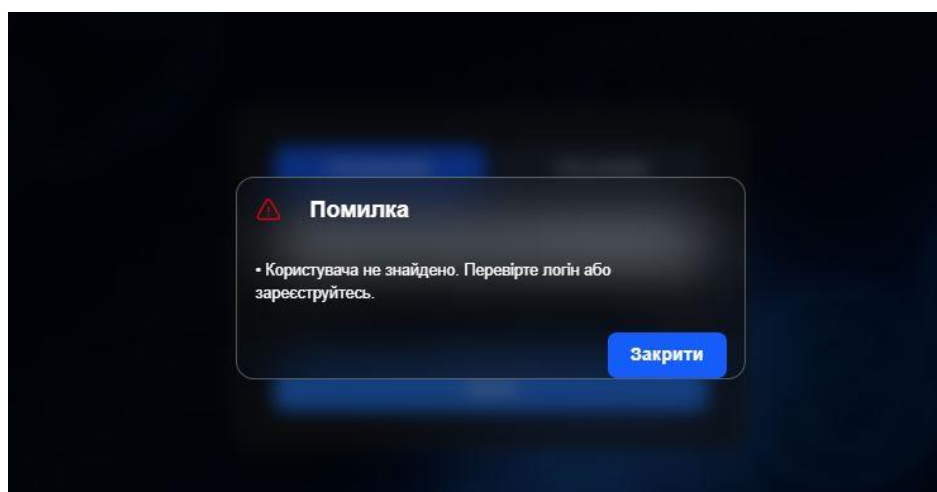


Рисунок 3.42 – Користувача не знайдено у системі

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				76
Змн.	Арк.	№ докум.	Підпис	Дата		

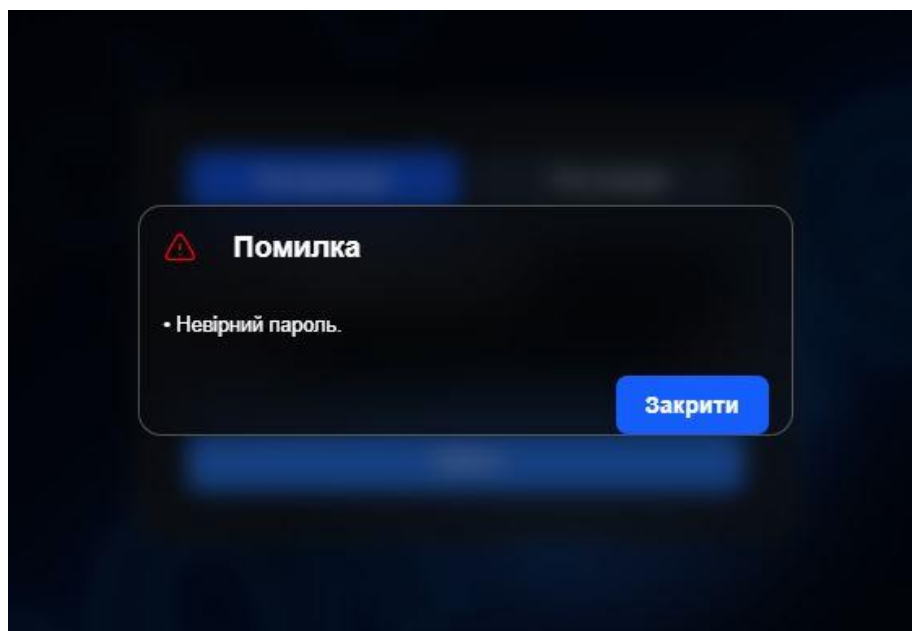


Рисунок 3.43 – Користувач ввів неправильний пароль

В системі реалізовано перевірку доступу до сторінок, у месенджер можуть заходити лише авторизовані користувачі, інших варіантів немає. Тобто, якщо користувач неавторизований, він не зможе перейти навіть на головну сторінку чатів, що видно на рис. 3.44. Аналогічна ситуація і з іншими сторінками, наприклад, зі сторінкою профілю, як показано на рис. 3.45.

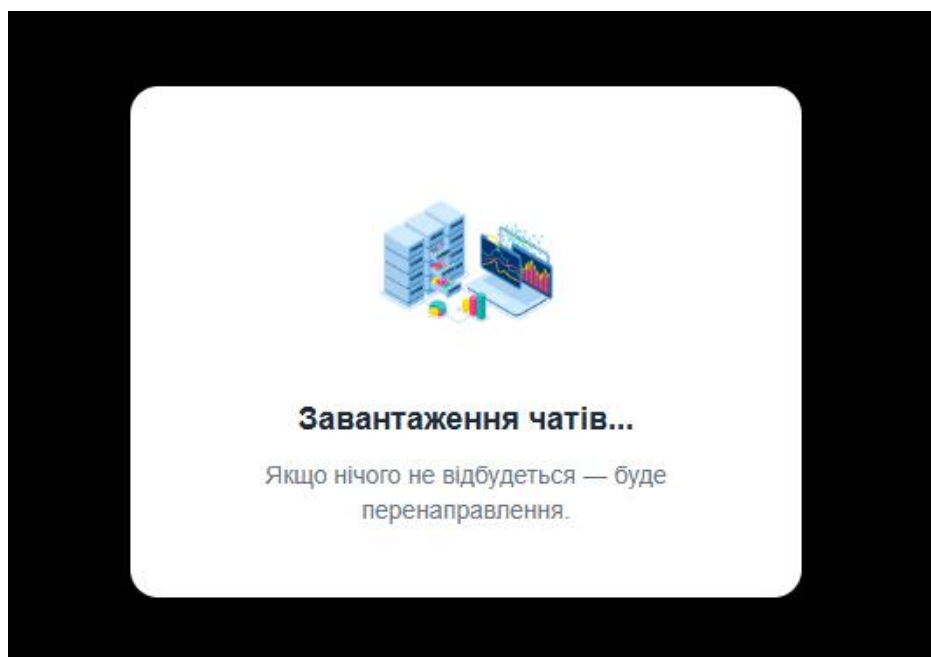


Рисунок 3.44 – Неавторизований користувач намагається потрапити на сторінку чатів

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

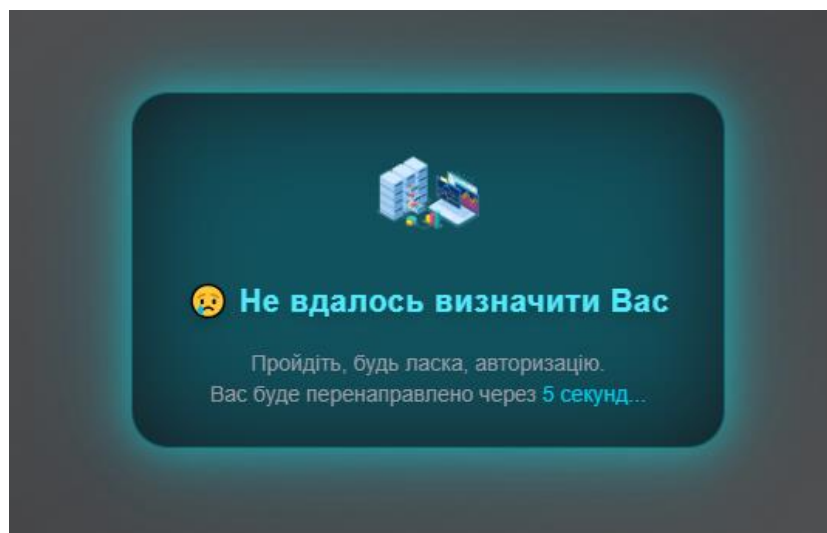


Рисунок 3.45 – Неавторизований користувач намагається потрапити на сторінку профілю

Ще одна потенційна проблема – це ситуація, коли користувач намагається потрапити до чата певної групи або до особистого чату, в якому він не є учасником. Тому потрібно забезпечити захист від входу користувачів, які не є учасниками групи або не авторизовані. Спочатку забороняємо доступ до групових чатів користувачам без авторизації, а після авторизації перевіряємо, чи дійсно користувач є учасником конкретної групи, як це видно на рис. 3.46.

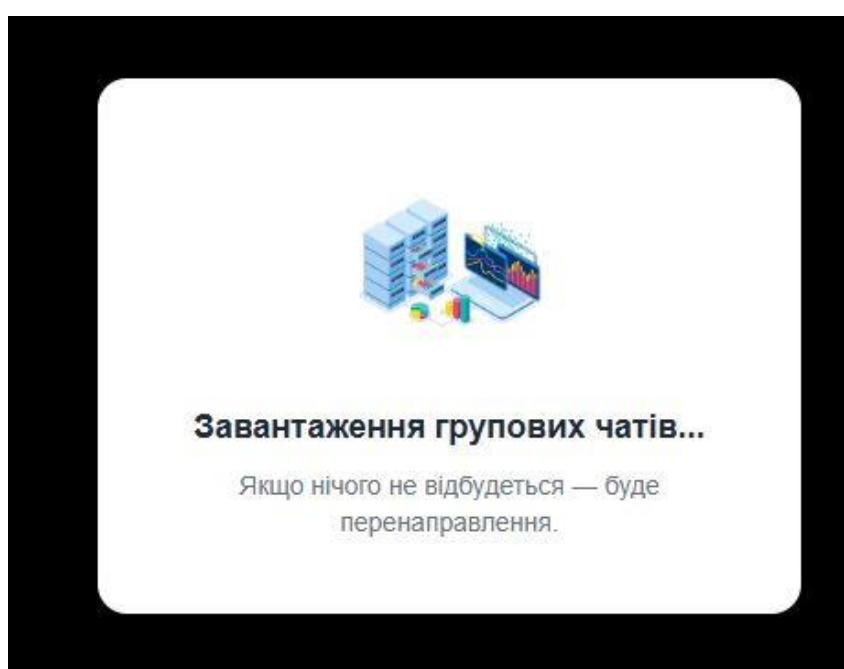


Рисунок 3.46 – Користувач намагається потрапити до групи, в якій його немає

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				78
Змн.	Арк.	№ докум.	Підпис	Дата		

Однією з важливих перевірок є захист доступу до особистого чату від неавторизованих користувачів. Особисте листування – переважно розмова двох осіб, тому необхідно забезпечити надійну безпеку, щоб сторонні потенційно небезпечні користувачі не могли отримати доступ до вмісту переписки. Для цього передбачено, що перш за все користувач має бути авторизованим, тому будь-який неавторизований користувач не зможе переглянути особисті листування, як це показано на рис. 3.47.

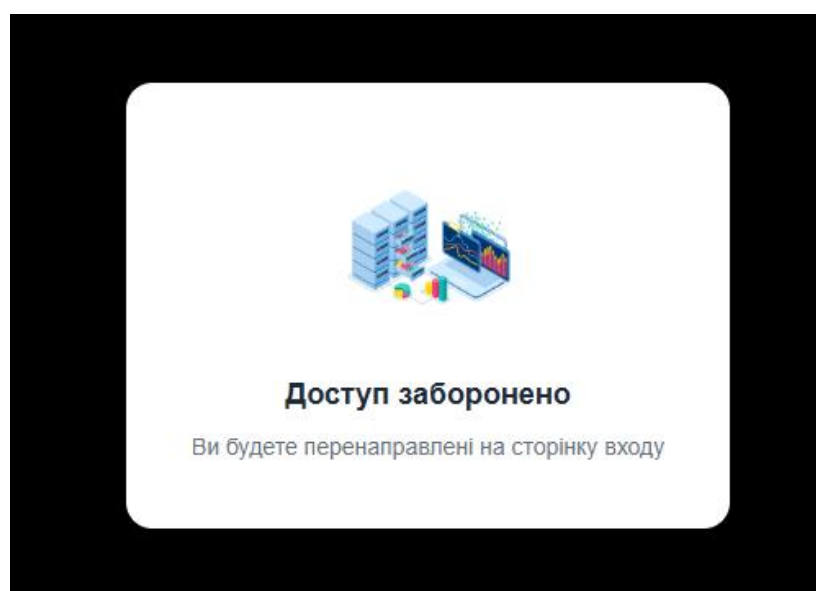


Рисунок 3.47 – Неавторизований користувач намагається потрапити на сторінку з чатом

Після завершення фази тестування можна зазначити, що вебмесенджер успішно пройшов усі зазначені перевірки та тести. Кожен компонент і метод працювали без помилок і виконали свої функції коректно.

Висновки до третього розділу

У результаті проведеної роботи було здійснено всебічний огляд інтерфейсу вебмесенджера, який включав у себе детальну характеристику кожної сторінки. Окрім того, було проведено тестування з метою виявлення потенційних помилок та розроблено алгоритми їх обробки.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У випускній кваліфікаційній роботі бакалавра розроблено веборієнтовану систему обміну повідомленнями, а саме вебмесенджер “Connect”.

У процесі підготовки до розробки було проаналізовано кілька популярних вебмесенджерів, зокрема їх функціональні можливості, архітектурні рішення, переваги та недоліки. На основі цього аналізу сформульовано функціональні та нефункціональні вимоги до майбутньої системи. Для реалізації обрано архітектуру MVC, яка забезпечує чіткі розділення відповідальностей між компонентами системи. Як основні засоби розробки використано сучасний технологічний стек, Next.js було обрано для розробки фронтенду, NestJS для розробки серверної частини, PostgreSQL для бази даних, також були використані сторонні сервіси Twilio та Google API для підтримки реєстрації, автентифікації та сповіщень.

Було представлено та детально описано загальний алгоритм роботи вебмесенджера, з урахуванням логіки обробки повідомлень, автентифікації користувачів та управління групами. Також розроблено структуру бази даних, яка складається з 12 взаємопов'язаних таблиць, що забезпечують зберігання користувацьких даних, повідомлень, медіафайлів, інформації про групи та інші важливі сутності. Окрім цього було розглянуто діаграму класів, яка відображає основні компоненти системи, їхні атрибути, методи та взаємозв'язки, із детальним поясненням кожного класу в архітектурі застосунку.

Підводячи підсумки, можна зазначити, що було розроблено повноцінний інтерфейс вебмесенджера, який відповідає встановленим вимогам щодо зручності, адаптивності та інтуїтивної навігації. Інтерфейс реалізовано українською мовою, з підтримкою сучасного дизайну та інтерактивних елементів для комфортної взаємодії користувачів.

Система успішно пройшла кілька етапів тестування основних функціональних модулів, зокрема реєстрації та входу користувачів, відправлення та отримання повідомлень у приватних і групових чатах, розмежування прав доступу, обробки вкладень, а також функцій пошуку, створення та управління групами.

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				80
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Архітектура сучасних вебмесенджерів [Електронний ресурс] – Режим доступу до ресурсу: <https://ceur-ws.org>
2. Керест Н.І., Локтікова Т.М., Лисогор Ю.І. PWA вебмесенджер з покращеною автентифікацією та функцією обміну контентом. Тези доповідей Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 12 травня 2025 року. Подані та прийняті до опублікування.
3. Керест Н.І., Локтікова Т.М., Петросян Р.В. Аналіз ключових вимог та особливостей реалізації веб-месенджера. Тези доповідей VII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», 02–03 грудня 2024 року. Житомир: Житомирська політехніка, 2024. – С. 67–68.
4. Лорето С., Романо С. Комунікація в реальному часі з WebRTC / Сальваторе Лорето, Саймон П'єтро Романо. – O'Reilly, 2014. – 150 с.
5. Сінгх А. Розробка корпоративного чат-бота: використання відкритого програмного забезпечення / Абхішек Сінгх. – Apress, 2019. – 368 с.
6. Шеват А. Проектування ботів: створення розмовних інтерфейсів / Амір Шеват. – O'Reilly Media, 2017. – 320 с.
7. cloudinary.com [Електронний ресурс] – Режим доступу до ресурсу: <https://cloudinary.com/documentation>
8. developer.mozilla.org – WebSocket API [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API
9. [docker.com](https://docs.docker.com) [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com>
10. jwt.io [Електронний ресурс] – Режим доступу до ресурсу: <https://jwt.io/introduction>
11. [messenger.com](https://www.messenger.com) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.messenger.com>

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова, Т. М.				81
Змн.	Арк.	№ докум.	Підпис	Дата		

12. NestJS.com [Електронний ресурс] – Режим доступу до ресурсу: <https://nestjs.com>
13. Nextjs.org [Електронний ресурс] – Режим доступу до ресурсу: <https://nextjs.org>
14. OWASP.org – Безпека вебзастосунків [Електронний ресурс] – Режим доступу до ресурсу: <https://owasp.org>
15. pgadmin.org [Електронний ресурс] – Режим доступу до ресурсу: <https://www.pgadmin.org>
16. postgresql.org [Електронний ресурс] – Режим доступу до ресурсу: <https://www.postgresql.org>
17. postman.com [Електронний ресурс] – Режим доступу до ресурсу: <https://learning.postman.com>
18. prisma.io [Електронний ресурс] – Режим доступу до ресурсу: <https://www.prisma.io>
19. Signal Foundation. Протокол шифрування Signal [Електронний ресурс] – Режим доступу до ресурсу: <https://signal.org/docs/>
20. slack.com [Електронний ресурс] – Режим доступу до ресурсу: <https://slack.com/features>
21. tailwindcss.com [Електронний ресурс] – Режим доступу до ресурсу: <https://tailwindcss.com>
22. teams.microsoft.com [Електронний ресурс] – Режим доступу до ресурсу: <https://www.microsoft.com/microsoft-teams>
23. telegram.org [Електронний ресурс] – Режим доступу до ресурсу: <https://web.telegram.org>
24. twilio.com [Електронний ресурс] – Режим доступу до ресурсу: <https://www.twilio.com/docs>
25. web.whatsapp.com [Електронний ресурс] – Режим доступу до ресурсу: <https://web.whatsapp.com>

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова, Т. М.				82
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Керест Н. І.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Локтікова. Т. М.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

Вебсокет файл з управлінням особистих користувацьких чатів MessageGateway

```
import {
  WebSocketGateway,
  SubscribeMessage,
  MessageBody,
  ConnectedSocket,
  OnGatewayConnection,
  OnGatewayDisconnect,
  WebSocketServer,
  WsException
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { MessageService } from '../message.service';
import { UserService } from '../user/user.service';
import { JwtAuthGuard } from '../auth/jwt-auth.guard';
import { JwtService } from '@nestjs/jwt';
import { UseGuards } from '@nestjs/common';
import { CreateFolderDto } from 'src/files/dto/create-folder.dto';
import { CreateFileDto } from 'src/files/dto/create-file.dto';
import { FilesService } from 'src/files/files.service';
import { UpdateFileDto } from 'src/files/dto/update-file.dto';
import { RequestAccessDto } from 'src/files/dto/request-access.dto';
import { ShareAccessDto } from 'src/files/dto/share-access.dto';
import { AccessRequest } from '../files/entities/access-request.entity';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';

@WebSocketGateway({
  cors: {
    origin: '*',
  },
})
export class MessageGateway implements OnGatewayConnection,
OnGatewayDisconnect {
  [x: string]: any;
  public onlineUsers: Map<number, string> = new Map();
  @WebSocketServer()
  server: Server;
  constructor(
    @InjectRepository(AccessRequest)
    private readonly accessRequestRepository: Repository<AccessRequest>,
    private readonly messageService: MessageService,
    private readonly jwtService: JwtService,
```

```

    private readonly filesService: FilesService,
    private readonly userService: UserService,
  ) {}

  handleConnection(client: Socket) {
    try {
      const token = client.handshake.auth?.token ||
client.handshake.headers?.authorization?.split(' ')[1];

      if (!token) {
        console.warn('[WS] Missing token');
        client.disconnect();
        return;
      }

      const decoded = this.jwtService.decode(token) as any;
      const userId = decoded?.sub;

      if (userId) {
        const existingSocketId = this.onlineUsers.get(userId);
        if (existingSocketId && existingSocketId !== client.id) {
          const existingClient = this.server.sockets.sockets.get(existingSocketId);
          existingClient?.disconnect(true);
        }

        this.onlineUsers.set(userId, client.id);
        this.server.emit('userOnline', { userId });

        this.userService.updateLastSeen(userId, new Date())
      } else {
        client.disconnect();
      }
    } catch (err) {
      client.disconnect();
    }
  }

  handleDisconnect(client: Socket) {
    const userId = [...this.onlineUsers.entries()].find(([ socketId ]) => socketId ===
client.id)?.[0];
    if (userId) {
      this.onlineUsers.delete(userId);
      this.server.emit('userOffline', { userId });

      this.userService.updateLastSeen(userId, new Date())
    }
  }

```

```

    }

    @SubscribeMessage('getMyChats')
    async handleGetMyChats(
        @MessageBody() data: { userId: number },
        @ConnectedSocket() client: Socket,
    ) {
        try {
            const chats = await this.messagesService.getUserChats(data.userId);
            this.server.to(client.id).emit('myChats', chats);
        } catch (err) {
            this.server.to(client.id).emit('error', 'Не вдалося отримати чати');
        }
    }

    @SubscribeMessage('sendMessage')
    async handleSendMessage(
        @MessageBody() data: {
            senderId: number;
            receiverId: number;
            content?: string;
            fileUrl?: string;
        },
        @ConnectedSocket() client: Socket,
    ) {
        const { senderId, receiverId, content = '', fileUrl } = data;

        const message = await this.messagesService.createMessage(senderId, receiverId,
            content, fileUrl);

        const chatsSender = await this.messagesService.getUserChats(senderId);
        const chatsReceiver = await this.messagesService.getUserChats(receiverId);

        const receiverSocketId = this.onlineUsers.get(receiverId);
        const senderSocketId = this.onlineUsers.get(senderId);

        if (receiverSocketId) {
            this.server.to(receiverSocketId).emit('newMessage', message);
            this.server.to(receiverSocketId).emit('myChats', chatsReceiver);
        }

        if (senderSocketId && senderSocketId !== client.id) {
            this.server.to(senderSocketId).emit('messageStatusUpdate', {
                messageId: message.id,
                status: 'sent',
            });
        }
    }

```

```

    }

    return message;
}

@SubscribeMessage('updateMessage')
async handleUpdateMessage(
  @MessageBody() data: { messageId: number; userId: number; content: string },
  @ConnectedSocket() client: Socket,
) {
    const updatedMessage = await
this.messagesService.updateMessage(data.messageId, data.userId, data.content);

    this.server.to(client.id).emit('messageUpdated', updatedMessage);
    const receiverSocketId = this.onlineUsers.get(updatedMessage.receiver.id);

    if (receiverSocketId) {
        this.server.to(receiverSocketId).emit('messageUpdated', updatedMessage);
    }
    return updatedMessage;
}

@SubscribeMessage('deleteMessage')
async handleDeleteMessage(
  @MessageBody() data: { messageId: number; userId: number },
  @ConnectedSocket() client: Socket,
) {
    const deletedMessage = await this.messagesService.deleteMessage(data.messageId,
data.userId);
    this.server.to(client.id).emit('messageDeleted', { messageId: data.messageId });
    const otherUserSocketId = this.onlineUsers.get(deletedMessage.receiver.id);
    if (otherUserSocketId) {
        this.server.to(otherUserSocketId).emit('messageDeleted', { messageId:
data.messageId });
    }
}

@SubscribeMessage('updateStatus')
async handleUpdateStatus(
  @MessageBody() data: { messageId: number; status: 'read' | 'delivered'; userId:
number },
  @ConnectedSocket() client: Socket,
) {
    try {

```

```

        const updated = await this.messagesService.updateStatus(data.messageId,
data.userId, data.status);
        this.server.emit('messageStatusUpdate', { messageId: updated.id, status:
updated.status });
    } catch (e) {
        console.error(e);
        throw new WsException(e.message);
    }
}

```

```

@SubscribeMessage('sendMessageWithFile')
async handleSendMessageWithFile(
    @MessageBody() data: { senderId: number; receiverId: number; content?: string;
fileUrl: string },
    @ConnectedSocket() client: Socket,
) {
    const message = await this.messagesService.sendMessageWithFile(
        data.senderId,
        data.receiverId,
        data.content || "",
        data.fileUrl,
    );

    const receiverSocketId = this.onlineUsers.get(data.receiverId);
    if (receiverSocketId) {
        this.server.to(receiverSocketId).emit('newMessage', message);
    }

    return message;
}

```

```

@SubscribeMessage('createFolder')
async handleCreateFolder(
    @MessageBody() dto: CreateFolderDto,
    @ConnectedSocket() client: Socket,
) {
    const folder = await this.filesService.createFolder(dto);
    this.server.emit('folderCreated', folder);
    return folder;
}

```

```

@SubscribeMessage('createFile')
async handleCreateFile(
    @MessageBody() dto: CreateFileDto,
    @ConnectedSocket() client: Socket,
) {

```

```

    const file = await this.filesService.createFile(dto);
    this.server.emit('fileCreated', file);
    return file;
}

@SubscribeMessage('updateFile')
async handleUpdateFile(
  @MessageBody() data: { fileId: number, userId: number, dto: UpdateFileDto },
) {
  const file = await this.filesService.updateFile(data.fileId, data.userId, data.dto);
  this.server.emit('fileUpdated', file);
  return file;
}

@SubscribeMessage('deleteFile')
async handleDeleteFile(
  @MessageBody() data: { fileId: number, userId: number },
) {
  await this.filesService.deleteFile(data.fileId, data.userId);
  this.server.emit('fileDeleted', { fileId: data.fileId });
}

@SubscribeMessage('requestAccess')
async handleRequestAccess(
  @MessageBody() dto: RequestAccessDto,
) {
  const request = await this.filesService.requestAccess(dto);
  const receiverSocketId = this.onlineUsers.get(dto.toUserId);
  if (receiverSocketId) {
    this.server.to(receiverSocketId).emit('accessRequested', request);
  }
  return request;
}

@SubscribeMessage('shareAccess')
async handleShareAccess(
  @MessageBody() dto: ShareAccessDto,
) {
  const result = await this.filesService.shareAccess(dto);
  const sharedWithSocket = this.onlineUsers.get(dto.userId);
  if (sharedWithSocket) {
    this.server.to(sharedWithSocket).emit('accessGranted', result);
  }
  return result;
}

```

```

@SubscribeMessage('approveAccessRequest')
async handleApproveAccessRequest(
  @MessageBody() data: { requestId: number, targetId: number, isFolder: boolean }
) {
  const request = await this.accessRequestRepository.findOne({
    where: { id: data.requestId },
    relations: ['fromUser', 'toUser', 'file', 'folder'],
  });

  if (!request || request.status !== 'pending') {
    throw new Error('Запит не знайдено або вже оброблено');
  }

  if (!data.targetId) {
    throw new Error('Не вказано ресурс для доступу');
  }
  if (data.isFolder) {
    request.folder = { id: data.targetId } as any;
    request.file = null;
  } else {
    request.file = { id: data.targetId } as any;
    request.folder = null;
  }

  request.status = 'approved';
  const savedRequest = await this.accessRequestRepository.save(request);
  const receiverSocketId = this.onlineUsers.get(savedRequest.fromUser.id);
  if (receiverSocketId) {
    this.server.to(receiverSocketId).emit('accessGranted', {
      fromUser: savedRequest.toUser,
      targetId: data.targetId,
      isFolder: data.isFolder,
      permission: savedRequest.permission,
    });
  }

  return savedRequest;
}

```

```

@SubscribeMessage('requestGeneralAccess')
async handleRequestGeneralAccess(
  @MessageBody() dto: { fromUserId: number, toUserId: number, permission: 'read'
    | 'write' },
) {
  const request = await this.filesService.requestGeneralAccess(dto);
}

```

```

const receiverSocketId = this.onlineUsers.get(dto.toUserId);

if (receiverSocketId) {
  this.server.to(receiverSocketId).emit('generalAccessRequested', request);
}

return request;
}

@SubscribeMessage('approveGeneralAccessRequest')
async handleApproveGeneralAccessRequest(
  @MessageBody() data: { requestId: number, targetId: number, isFolder: boolean },
) {
  const request = await this.filesService.approveAccessRequest(data.requestId);

  await this.filesService.shareAccess({
    userId: request.fromUser.id,
    targetId: data.targetId,
    isFolder: data.isFolder,
    permission: request.permission,
  });

  const receiverSocketId = this.onlineUsers.get(request.fromUser.id);
  if (receiverSocketId) {
    this.server.to(receiverSocketId).emit('accessGranted', {
      fromUser: request.toUser,
      targetId: data.targetId,
      isFolder: data.isFolder,
      permission: request.permission,
    });
  }
  return request;
}

@SubscribeMessage('getOnlineUsers')
handleGetOnlineUsers(@ConnectedSocket() client: Socket) {
  const onlineUserIds = [...this.onlineUsers.keys()];
  client.emit('onlineUsers', onlineUserIds);
}
}

```

Вебсокет файл з управлінням групових сторінок GroupChatGateway

```
import {
  WebSocketGateway,
  SubscribeMessage,
  MessageBody,
  ConnectedSocket,
  WebSocketServer,
  OnGatewayConnection,
  OnGatewayDisconnect,
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { GroupChatService } from '../group-chat.service';
import { JwtService } from '@nestjs/jwt';
import { ForbiddenException, NotFoundException } from '@nestjs/common';

@WebSocketGateway({ cors: { origin: '*' } })
export class GroupChatGateway implements OnGatewayConnection,
OnGatewayDisconnect {

  @WebSocketServer()
  server: Server;
  public onlineUsers: Map<number, string> = new Map();
  constructor(private readonly groupChatService: GroupChatService, private readonly
jwtService: JwtService,) { }
  handleConnection(client: Socket) {
    try {
      const token = client.handshake.auth?.token ||
client.handshake.headers?.authorization?.split(' ')[1];

      if (!token) {
        console.warn('[WS] Missing token');
        client.disconnect();
        return;
      }

      const decoded = this.jwtService.decode(token) as any;
      const userId = decoded?.sub;

      if (userId) {
        const existingSocketId = this.onlineUsers.get(userId);
        if (existingSocketId && existingSocketId !== client.id) {
          const existingClient = this.server.sockets.sockets.get(existingSocketId);
          existingClient?.disconnect(true);
        }
      }
    }
  }
}
```

```

        this.onlineUsers.set(userId, client.id);
        client.join(userId.toString());
    } else {
        client.disconnect();
    }
} catch (err) {
    console.error('[WS] Auth error', err);
    client.disconnect();
}
}

```

```

handleDisconnect(client: Socket) {
    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (userId) {
        this.onlineUsers.delete(userId);
    }
}

```

```

@SubscribeMessage('getGroupInfo')
async getGroupInfo(
    @MessageBody() data: { groupId: number; userId: number },
    @ConnectedSocket() client: Socket,
) {
    const groupChat = await this.groupChatService.getGroupById(data.groupId);
    if (!groupChat) {
        client.emit('error', { message: 'Group not found' });
        return;
    }
    client.emit('groupInfo', groupChat);
}

```

```

@SubscribeMessage('createGroup')
async createGroup(
    @MessageBody() data: { name: string; participantIds: number[]; userId: number },
    @ConnectedSocket() client: Socket,
) {
    const group = await this.groupChatService.createGroup(
        data.userId,
        data.name,
        data.participantIds,
    );
    group.participants.forEach((user) => {
        this.server.to(user.id.toString()).emit('groupCreated', group);
    });
}

```

```
}
```

```
@SubscribeMessage('getMyGroups')
async getMyGroups(
  @MessageBody() data: { userId: number },
  @ConnectedSocket() client: Socket,
) {
  const groups = await this.groupChatService.getGroupsForUser(data.userId);
  client.emit('myGroups', groups);
}
```

```
@SubscribeMessage('addParticipant')
async addParticipant(
  @MessageBody() data: { groupId: number; userNameToAdd: string; userId: number
},
  @ConnectedSocket() client: Socket,
) {
  const updatedGroup = await this.groupChatService.addParticipantByUsername(
    data.userId,
    data.groupId,
    data.userNameToAdd,
  );

  for (const user of updatedGroup.participants) {
    this.server.to(user.id.toString()).emit('participantAdded', updatedGroup);

    const userGroups = await this.groupChatService.getGroupsForUser(user.id);
    this.server.to(user.id.toString()).emit('myGroups', userGroups);
  }
}
```

```
@SubscribeMessage('removeParticipant')
async removeParticipant(
  @MessageBody() data: { groupId: number; userIdToRemove: number; userId:
number },
  @ConnectedSocket() client: Socket,
) {
  const updatedGroup = await this.groupChatService.removeParticipant(
    data.groupId,
    data.userIdToRemove,
  );

  const payload = {
    updatedGroup,
    initiatorId: data.userId,
  };
}
```

```

};

for (const user of updatedGroup.participants) {
  this.server.to(user.id.toString()).emit('participantRemoved', payload);
  const groups = await this.groupChatService.getGroupsForUser(user.id);
  this.server.to(user.id.toString()).emit('myGroups', groups);
}

this.server.to(data.userIdToRemove.toString()).emit('participantRemoved', payload);
const removedUserGroups = await
this.groupChatService.getGroupsForUser(data.userIdToRemove);
  this.server.to(data.userIdToRemove.toString()).emit('myGroups',
removedUserGroups);
}

@SubscribeMessage('renameGroup')
async renameGroup(
  @MessageBody() data: { groupId: number; name: string; userId: number },
  @ConnectedSocket() client: Socket,
) {
  const updatedGroup = await this.groupChatService.renameGroup(
    data.userId,
    data.groupId,
    data.name,
  );

  updatedGroup.participants.forEach((user) => {
    this.server.to(user.id.toString()).emit('groupRenamed', updatedGroup);
  });
}

@SubscribeMessage('toggleAdmin')
async toggleAdmin(
  @MessageBody() data: { groupId: number; targetUserId: number; actorUserId:
number },
  @ConnectedSocket() client: Socket,
) {
  const updatedGroup = await this.groupChatService.toggleAdminStatus(
    data.actorUserId,
    data.groupId,
    data.targetUserId,
  );

  updatedGroup.participants.forEach((user) => {
    this.server.to(user.id.toString()).emit('adminStatusChanged', updatedGroup);
  });
}

```

```

}

@SubscribeMessage('leaveGroup')
async handleLeaveGroup(
  @MessageBody() data: { groupId: number, userId: number },
  @ConnectedSocket() client: Socket
) {
  const updatedGroup = await
this.groupChatService.removeParticipant(data.groupId, data.userId);
  this.server.to(data.groupId.toString()).emit('participantRemoved', updatedGroup);
}

@SubscribeMessage('deleteGroup')
async handleDeleteGroup(
  @MessageBody() data: { groupId: number; userId: number },
  @ConnectedSocket() client: Socket
) {
  const group = await this.groupChatService.getGroupWithOwner(data.groupId);
  if (!group) throw new NotFoundException('Group not found');

  if (group.owner.id !== data.userId) {
    throw new ForbiddenException('Only the owner can delete the group');
  }

  this.server.to(`group_${data.groupId}`).emit('groupDeleted', {
    groupId: data.groupId,
    groupName: group.name,
  });

  client.emit('groupDeleted', {
    groupId: data.groupId,
    groupName: group.name,
  });

  await this.groupChatService.deleteGroup(data.groupId, data.userId);
}
}

```

Вебсокет файл з управлінням групових чатів GroupMessageGateway

```
import {
  WebSocketGateway,
  WebSocketServer,
  SubscribeMessage,
  MessageBody,
  ConnectedSocket,
  OnGatewayConnection,
  OnGatewayDisconnect,
} from '@nestjs/websockets';
import { Server, Socket } from 'socket.io';
import { GroupMessageService } from '../group-message.service';
import { JwtService } from '@nestjs/jwt';
import { extname, join } from 'path';
import * as fs from 'fs';

@WebSocketGateway({ cors: { origin: '*' } })
export class GroupMessageGateway implements OnGatewayConnection,
OnGatewayDisconnect {
  @WebSocketServer()
  server: Server;
  public onlineUsers: Map<number, string> = new Map();

  constructor(
    private readonly groupMessageService: GroupMessageService,
    private readonly jwtService: JwtService,
  ) {}

  handleConnection(client: Socket) {
    try {
      const token = client.handshake.auth?.token ||
client.handshake.headers?.authorization?.split(' ')[1];
      if (!token) {
        console.warn('[WS] Missing token');
        client.disconnect();
        return;
      }
      const decoded = this.jwtService.decode(token) as any;
      const userId = decoded?.sub;

      if (userId) {
        const existingSocketId = this.onlineUsers.get(userId);
```

```

        if (existingSocketId && existingSocketId !== client.id) {
            const existingClient = this.server.sockets.sockets.get(existingSocketId);
            existingClient?.disconnect(true);
        }
        this.onlineUsers.set(userId, client.id);
        client.join(userId.toString());
    } else {
        client.disconnect();
    }
} catch (err) {
    console.error('[WS] Auth error', err);
    client.disconnect();
}
}
handleDisconnect(client: Socket) {
    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (userId) {
        this.onlineUsers.delete(userId);
    }
}
}
@SubscribeMessage('sendGroupMessage')
async handleSendGroupMessage(
    @MessageBody() data: {
        groupId: number;
        content?: string;
        file?: {
            name: string;
            type: string;
            base64: string;
        };
    },
    @ConnectedSocket() client: Socket,
) {
    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (!userId) return;
    let savedFile: Express.Multer.File | null = null;
    let fileUrl: string | undefined;
    if (data.file) {
        const buffer = Buffer.from(data.file.base64, 'base64');
        const uploadsDir = join(process.cwd(), 'uploads');
        await fs.promises.mkdir(uploadsDir, { recursive: true });
        const ext = extname(data.file.name);
        const filename = `${Date.now()}-${Math.round(Math.random() * 1e9)}${ext}`;
        const filePath = join(uploadsDir, filename);

```

```

await fs.promises.writeFile(filePath, buffer);
savedFile = {
  originalname: data.file.name,
  mimetype: data.file.type,
  filename,
  path: filePath,
  size: buffer.length,
  buffer,
  fieldname: 'file',
  destination: uploadsDir,
  encoding: '7bit',
  stream: null as any,
} as Express.Multer.File;

fileUrl = '/uploads/' + filename;
}
const message = await this.groupMessageService.sendMessage(
  data.groupId,
  userId,
  data.content,
  fileUrl,
);
this.server.to(`group_${data.groupId}`).emit('groupMessageSent', message);
}

@SubscribeMessage('getGroupMessages')
async handleGetMessages(
  @MessageBody() data: { groupId: number },
  @ConnectedSocket() client: Socket,
) {
  const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId === client.id)?.[0];
  if (!userId) return;

  const messages = await this.groupMessageService.getMessages(data.groupId);
  client.emit('groupMessages', messages);
}

@SubscribeMessage('updateGroupMessage')
async handleUpdateMessage(
  @MessageBody() data: { messageId: number; content: string },
  @ConnectedSocket() client: Socket,
) {
  const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId === client.id)?.[0];
  if (!userId) return;

```

```

    const updatedMessage = await this.groupMessageService.update(data.messageId,
data.content, userId);
    this.server.emit('groupMessageUpdated', updatedMessage);
  }
  @SubscribeMessage('deleteGroupMessage')
  async handleDeleteMessage(
    @MessageBody() data: { messageId: number },
    @ConnectedSocket() client: Socket,
  ) {
    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (!userId) return;
    await this.groupMessageService.delete(data.messageId, userId);
    this.server.emit('groupMessageDeleted', { id: data.messageId });
  }
  @SubscribeMessage('joinGroup')
  handleJoinGroup(
    @MessageBody() data: { groupId: number },
    @ConnectedSocket() client: Socket,
  ) {
    client.join(`group_${data.groupId}`);
  }
  @SubscribeMessage('markGroupMessagesAsRead')
  async handleMarkMultipleAsRead(
    @MessageBody() data: { messageIds: number[] },
    @ConnectedSocket() client: Socket
  ) {
    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (!userId) return;
    const updates = await Promise.all(
      data.messageIds.map(async (id) => {
        const readers = await this.groupMessageService.markAsRead(id, userId);
        return { id, readers };
      })
    );
    updates.forEach(({ id, readers }) => {
      this.server.emit('groupMessageReadUpdate', {
        messageId: id,
        readers,
      });
    });
  }
}

```

Сервіс для роботи з віртуальною файловою системою `FilesService`
`@Injectable()`
`export class FilesService implements OnModuleInit {`

```

    [x: string]: any;
    constructor(
        @InjectRepository(File) private fileRepo: Repository<File>,
        @InjectRepository(Folder) private folderRepo: Repository<Folder>,
        @InjectRepository(FileAccess) private accessRepo: Repository<FileAccess>,
        @InjectRepository(AccessRequest) private accessRequestRepository:
Repository<AccessRequest>,
        @InjectRepository(User) private readonly userRepository: Repository<User>,
    ) {}
    async onModuleInit() {
        await this.createFoldersForExistingUsers();
    }

    async createFolder(dto: CreateFolderDto) {
        const folder = this.folderRepo.create({
            name: dto.name,
            owner: { id: dto.ownerId },
            ...(dto.parentId && { parent: { id: dto.parentId } }),
        });

        return this.folderRepo.save(folder);
    }

    async createFile(dto: CreateFileDto) {
        const file = this.fileRepo.create({
            name: dto.name,
            url: dto.url,
            owner: { id: dto.ownerId },
            ...(dto.parentFolderId && { folder: { id: dto.parentFolderId } }),
        });

        return this.fileRepo.save(file);
    }

    async shareAccess(dto: ShareAccessDto) {
        const existing = await this.accessRepo.findOne({
            where: {
                user: { id: dto.userId },
                ...(dto.isFolder
                    ? { folder: { id: dto.targetId } }
                    : { file: { id: dto.targetId } }),
            },
        });
    }

```

```

    },
  });

  if (existing) {
    if (existing.permission !== dto.permission) {
      existing.permission = dto.permission;
      return this.accessRepo.save(existing);
    }
    return existing;
  }
  const access = this.accessRepo.create({
    user: { id: dto.userId },
    ...(dto.isFolder
      ? { folder: { id: dto.targetId } }
      : { file: { id: dto.targetId } }),
    permission: dto.permission,
  });

  return this.accessRepo.save(access);
}

async getUserAccess(userId: number) {
  return this.accessRepo.find({
    where: { user: { id: userId } },
    relations: ['file', 'folder'],
  });
}

async getFolderContent(folderId: number | null, userId: number) {
  const whereFolder = folderId === null ? IsNull() : Equal(folderId);

  const files = await this.fileRepo.find({
    where: {
      folder: whereFolder,
      owner: { id: userId },
    },
    relations: ['owner'],
  });

  const folders = await this.folderRepo.find({
    where: {
      parent: whereFolder,
      owner: { id: userId },
    },
    relations: ['owner'],
  });

```

```

    return { folders, files };
}

async updateFile(fileId: number, userId: number, dto: UpdateFileDto) {
    const file = await this.fileRepo.findOne({
        where: { id: fileId },
        relations: ['owner'],
    });
    if (!file) throw new NotFoundException('Файл не знайдено');
    if (file.owner.id !== userId) throw new ForbiddenException('Немає доступу');

    if (dto.name) file.name = dto.name;
    if (dto.url) file.url = dto.url;

    if (dto.parentFolderId) {
        const parentFolder = await this.folderRepo.preload({ id: dto.parentFolderId });
        if (!parentFolder) throw new NotFoundException('Батьківська папка не знайдена');
        file.folder = parentFolder;
    }

    return this.fileRepo.save(file);
}

async deleteFile(fileId: number, userId: number) {
    const file = await this.fileRepo.findOne({
        where: { id: fileId },
        relations: ['owner'],
    });
    if (!file) throw new NotFoundException('Файл не знайдено');
    if (file.owner.id !== userId) throw new ForbiddenException('Немає доступу');

    return this.fileRepo.remove(file);
}

async updateFolder(folderId: number, userId: number, dto: UpdateFolderDto) {
    const folder = await this.folderRepo.findOne({
        where: { id: folderId },
        relations: ['owner'],
    });
    if (!folder) throw new NotFoundException('Папку не знайдено');
    if (folder.owner.id !== userId) throw new ForbiddenException('Немає доступу');

    if (dto.name) folder.name = dto.name;

```

```

    if (dto.parentId) {
        const parentFolder = await this.folderRepo.preload({ id: dto.parentId });
        if (!parentFolder) throw new NotFoundException('Батьківська папка не знайдена');
        folder.parent = parentFolder;
    }

    return this.folderRepo.save(folder);
}

async deleteFolder(folderId: number, userId: number) {
    const folder = await this.folderRepo.findOne({
        where: { id: folderId },
        relations: ['owner'],
    });
    if (!folder) throw new NotFoundException('Папку не знайдено');
    if (folder.owner.id !== userId) throw new ForbiddenException('Немає доступу');

    return this.folderRepo.remove(folder);
}

async requestAccess(dto: RequestAccessDto): Promise<AccessRequest> {
    const request = this.accessRequestRepository.create({
        fromUser: { id: dto.fromUserId },
        toUser: { id: dto.toUserId },
        permission: dto.permission,
        status: 'pending',
        ...(dto.isFolder && dto.targetId ? { folder: { id: dto.targetId } } :
            dto.targetId ? { file: { id: dto.targetId } } : {}),
    });

    return this.accessRequestRepository.save(request);
}

async getFileById(fileId: number) {
    return this.fileRepo.findOne({
        where: { id: fileId },
        relations: ['owner'],
    });
}

async approveAccessRequest(requestId: number): Promise<AccessRequest> {
    const request = await this.accessRequestRepository.findOne({
        where: { id: requestId },
        relations: ['fromUser', 'toUser', 'file', 'folder'],
    });

```

```

});

if (!request || request.status !== 'pending') {
    throw new Error('Запит не знайдено або вже оброблено');
}

request.status = 'approved';
return this.accessRequestRepository.save(request);
}

async requestGeneralAccess(dto: {
    fromUserId: number,
    toUserId: number,
    permission: 'read' | 'write',
}): Promise<AccessRequest> {
    const request: DeepPartial<AccessRequest> = {
        fromUser: { id: dto.fromUserId } as any,
        toUser: { id: dto.toUserId } as any,
        permission: dto.permission,
        status: 'pending',
    };

    const newRequest = this.accessRequestRepository.create(request);
    return this.accessRequestRepository.save(newRequest);
}

async createRootFolder(user: User): Promise<void> {
    const root = this.folderRepo.create({
        name: 'Root',
        owner: user,
        parent: null,
    });
    const savedRoot = await this.folderRepo.save(root);

    const structure = [
        {
            name: 'Documents',
            files: ['resume.docx', 'project.pdf'],
        },
        {
            name: 'Pictures',
            files: ['photo.jpg', 'screenshot.png'],
        },
        {
            name: 'Music',
            files: ['track.mp3', 'album.zip'],
        },
    ];

```

```

    },
  ];

  for (const folder of structure) {
    const subfolder = this.folderRepo.create({
      name: folder.name,
      owner: user,
      parent: savedRoot,
    });
    const savedSubfolder = await this.folderRepo.save(subfolder);
    for (const fileName of folder.files) {
      const file = this.fileRepo.create({
        name: fileName,
        owner: user,
        folder: savedSubfolder,
        url: `https://example.com/${fileName}`,
      });
      await this.fileRepo.save(file);
    }
  }

  const rootFiles = ['welcome.txt', 'readme.md'];
  for (const fileName of rootFiles) {
    const file = this.fileRepo.create({
      name: fileName,
      owner: user,
      folder: savedRoot,
      url: `https://example.com/${fileName}`,
    });
    await this.fileRepo.save(file);
  }
}

async createFoldersForExistingUsers() {
  const users = await this.userRepository.find();

  for (const user of users) {
    const rootFolder = await this.folderRepo.findOne({
      where: { owner: user, parent: IsNull() },
    });

    if (!rootFolder) {
      await this.createRootFolder(user);
    }
  }
}

```

```

async getPermission(
  userId: number,
  targetId: number | null,
  isFolder: boolean
): Promise<'read' | 'write' | null> {
  const access = await this.accessRepository.findOne({
    where: {
      user: { id: userId },
      ...(isFolder ? { folder: { id: targetId } } : { file: { id: targetId } })
    }
  });
  return access ? access.permission : null;
}

async updateMimeType() {
  const files = await this.fileRepo.find({ where: { mimeType: 'unknown' } });

  const updatedFiles: { id: number; name: string; mimeType: string }[] = [];

  for (const file of files) {
    try {
      const response = await fetch(file.url, { method: 'HEAD' });

      let mimeType = "";
      if (response.ok) {
        mimeType = response.headers.get('content-type') || "";
      } else {
        console.warn(`HEAD request failed for ${file.url}, status:
${response.status}`);
      }

      if (!mimeType || mimeType.startsWith('text/html')) {
        mimeType = mime.lookup(file.name) || 'application/octet-stream';
      }

      mimeType = mimeType.split(';')[0];

      file.mimeType = mimeType;
      await this.fileRepo.save(file);

      updatedFiles.push({ id: file.id, name: file.name, mimeType });
    } catch (error) {
      console.error(` ❌ Failed to fetch MIME type for file: ${file.name}`,
error.message);
    }
  }
}

```

```

    }
    return updatedFiles;
  }
  async getFileWithAccess(fileId: number): Promise<File> {
    return this.fileRepository.findOne({
      where: { id: fileId },
      relations: ['owner', 'folder', 'folder.owner'],
    });
  }
  async userHasAccessToFile(userId: number, fileId: number): Promise<boolean> {
    const access = await this.accessRepo.findOne({
      where: {
        user: { id: userId },
        file: { id: fileId },
      },
    });
    return !!access;
  }
  async getAccessibleFile(fileId: number, userId: number): Promise<File | null> {
    return this.fileRepo
      .createQueryBuilder('file')
      .leftJoinAndSelect('file.owner', 'owner')
      .leftJoin('file_access', 'access', 'access.fileId = file.id AND access.userId = :userId',
        { userId })
      .where('file.id = :fileId', { fileId })
      .andWhere(new Brackets(qb => {
        qb.where('owner.id = :userId', { userId }).orWhere('access.id IS NOT NULL');
      }))
      .getOne();
  }
  async getFileIfAccessible(fileId: number, userId: number): Promise<File | null> {
    return this.fileRepo
      .createQueryBuilder('file')
      .leftJoinAndSelect('file.owner', 'owner')
      .leftJoin('file_access', 'access', 'access.fileId = file.id AND access.userId = :userId', {
        userId })
      .where('file.id = :fileId', { fileId })
      .andWhere(new Brackets(qb => {
        qb.where('owner.id = :userId', { userId })
          .orWhere('access.id IS NOT NULL');
      }))
      .getOne();
  }
}

```

```

Сервіс та сокет для підключення штучного інтелекту до вебмесенджера
@WebSocketGateway({ cors: { origin: '*' } })
export class GptGateway implements OnGatewayConnection, OnGatewayDisconnect {
  @WebSocketServer()
  server: Server;
  public onlineUsers: Map<number, string> = new Map();

  constructor(
    private readonly gptService: GptService,
    private readonly jwtService: JwtService,
  ) {}

  handleConnection(client: Socket) {
    try {
      const token = client.handshake.auth?.token ||
        client.handshake.headers?.authorization?.split(' ')[1];

      if (!token) {
        console.warn('[WS] Missing token');
        client.disconnect();
        return;
      }

      const decoded = this.jwtService.decode(token) as any;
      const userId = decoded?.sub;

      if (userId) {
        const existingSocketId = this.onlineUsers.get(userId);
        if (existingSocketId && existingSocketId !== client.id) {
          const existingClient = this.server.sockets.sockets.get(existingSocketId);
          existingClient?.disconnect(true);
        }

        this.onlineUsers.set(userId, client.id);
        client.join(userId.toString());
      } else {
        client.disconnect();
      }
    } catch (err) {
      console.error('[WS] Auth error', err);
      client.disconnect();
    }
  }

  handleDisconnect(client: Socket) {

```

```

    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (userId) {
        this.onlineUsers.delete(userId);
    }
}

@SubscribeMessage('gpt:ask')
async handleMessage(
    @MessageBody() data: { message: string },
    @ConnectedSocket() client: Socket,
) {
    const answer = await this.gptService.ask(data.message);
    client.emit('gpt:response', { answer });
}
}

@Injectable()
export class GptService {
    private readonly endpoint = 'https://openrouter.ai/api/v1/chat/completions';
    private readonly apiKey = process.env.OPENROUTER_API_KEY;

    async ask(message: string): Promise<string> {
        try {
            const response = await axios.post(
                this.endpoint,
                {
                    model: 'mistralai/mistral-7b-instruct:free',
                    messages: [
                        { role: 'system', content: 'Ти дружелюбний український чат-бот. Відповідай
ввічливо, стисло і зрозуміло.' },
                        { role: 'user', content: message },
                    ],
                },
                {
                    headers: {
                        Authorization: `Bearer ${this.apiKey}`,
                        'Content-Type': 'application/json',
                        'HTTP-Referer': 'https://your-app.example.com',
                        'X-Title': 'Connect Messenger Chat',
                    },
                },
            );
            const answer = response.data?.choices?.[0]?.message?.content;

```

```

    if (answer) return answer;

    throw new Error('Invalid response from OpenRouter');
  } catch (e) {
    console.error('GPT error:', e.response?.data || e.message);
    throw new HttpException('OpenRouter API error', 500);
  }
}
}
}

```

Додаток Д

Сервіс для роботи з груповими чатами

```

@WebSocketGateway({ cors: { origin: '*' } })
export class GroupMessageGateway implements OnGatewayConnection,
OnGatewayDisconnect {
  @WebSocketServer()
  server: Server;
  public onlineUsers: Map<number, string> = new Map();

  constructor(
    private readonly groupMessageService: GroupMessageService,
    private readonly jwtService: JwtService,
  ) {}

  handleConnection(client: Socket) {
    try {
      const token = client.handshake.auth?.token ||
client.handshake.headers?.authorization?.split(' ')[1];

      if (!token) {
        console.warn('[WS] Missing token');
        client.disconnect();
        return;
      }

      const decoded = this.jwtService.decode(token) as any;
      const userId = decoded?.sub;

      if (userId) {
        const existingSocketId = this.onlineUsers.get(userId);
        if (existingSocketId && existingSocketId !== client.id) {
          const existingClient = this.server.sockets.sockets.get(existingSocketId);
          existingClient?.disconnect(true);
        }
      }
    }
  }
}

```

```

        this.onlineUsers.set(userId, client.id);
        client.join(userId.toString());
    } else {
        client.disconnect();
    }
} catch (err) {
    console.error('[WS] Auth error', err);
    client.disconnect();
}
}

handleDisconnect(client: Socket) {
    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (userId) {
        this.onlineUsers.delete(userId);
    }
}

@SubscribeMessage('sendGroupMessage')
async handleSendGroupMessage(
    @MessageBody() data: {
        groupId: number;
        content?: string;
        file?: {
            name: string;
            type: string;
            base64: string;
        };
    },
    @ConnectedSocket() client: Socket,
) {
    const userId = [...this.onlineUsers.entries()].find(([, socketId]) => socketId ===
client.id)?.[0];
    if (!userId) return;

    let savedFile: Express.Multer.File | null = null;
    let fileUrl: string | undefined;

    if (data.file) {
        const buffer = Buffer.from(data.file.base64, 'base64');
        const uploadsDir = join(process.cwd(), 'uploads');
        await fs.promises.mkdir(uploadsDir, { recursive: true });

        const ext = extname(data.file.name);
        const filename = `${Date.now()}-${Math.round(Math.random() * 1e9)}${ext}`;

```

```

const filePath = join(uploadsDir, filename);
await fs.promises.writeFile(filePath, buffer);

savedFile = {
  originalname: data.file.name,
  mimetype: data.file.type,
  filename,
  path: filePath,
  size: buffer.length,
  buffer,
  fieldname: 'file',
  destination: uploadsDir,
  encoding: '7bit',
  stream: null as any,
} as Express.Multer.File;

fileUrl = '/uploads/' + filename;
}

const message = await this.groupMessageService.sendMessage(
  data.groupId,
  userId,
  data.content,
  fileUrl,
);

this.server.to(`group_${data.groupId}`).emit('groupMessageSent', message);
}

@SubscribeMessage('getGroupMessages')
async handleGetMessages(
  @MessageBody() data: { groupId: number },
  @ConnectedSocket() client: Socket,
) {
  const userId = [...this.onlineUsers.entries()].find(([ , socketId]) => socketId === client.id)?.[0];
  if (!userId) return;
  const messages = await this.groupMessageService.getMessages(data.groupId);
  client.emit('groupMessages', messages);
}

@SubscribeMessage('updateGroupMessage')
async handleUpdateMessage(
  @MessageBody() data: { messageId: number; content: string },
  @ConnectedSocket() client: Socket,
) {
  const userId = [...this.onlineUsers.entries()].find(([ , socketId]) => socketId === client.id)?.[0];

```

```

    if (!userId) return;
    const updatedMessage = await this.groupMessageService.update(data.messageId,
data.content, userId);
    this.server.emit('groupMessageUpdated', updatedMessage);
  }
  @SubscribeMessage('deleteGroupMessage')
  async handleDeleteMessage(
    @MessageBody() data: { messageId: number },
    @ConnectedSocket() client: Socket,
  ) {
    const userId = [...this.onlineUsers.entries()].find(([ socketId ]) => socketId ===
client.id)?.[0];
    if (!userId) return;
    await this.groupMessageService.delete(data.messageId, userId);
    this.server.emit('groupMessageDeleted', { id: data.messageId });
  }
  @SubscribeMessage('joinGroup')
  handleJoinGroup(
    @MessageBody() data: { groupId: number },
    @ConnectedSocket() client: Socket,
  ) {
    client.join(`group_${data.groupId}`);
  }
  @SubscribeMessage('markGroupMessagesAsRead')
  async handleMarkMultipleAsRead(
    @MessageBody() data: { messageIds: number[] },
    @ConnectedSocket() client: Socket
  ) {
    const userId = [...this.onlineUsers.entries()].find(([ socketId ]) => socketId ===
client.id)?.[0];
    if (!userId) return;
    const updates = await Promise.all(
      data.messageIds.map(async (id) => {
        const readers = await this.groupMessageService.markAsRead(id, userId);
        return { id, readers };
      })
    );
    updates.forEach(({ id, readers }) => {
      this.server.emit('groupMessageReadUpdate', {
        messageId: id,
        readers,
      });
    });
  }
}

```

Сервіс для роботи з особистими повідомленнями

```
@Injectable()
export class MessageService {
  constructor(
    @InjectRepository(Message)
    private readonly messageRepository: Repository<Message>,

    private readonly userService: UserService,
  ) {}

  async sendMessage(senderId: number, receiverId: number, content: string):
  Promise<Message> {
    const message = this.messageRepository.create({
      sender: { id: senderId } as User,
      receiver: { id: receiverId } as User,
      content,
      status: 'sent',
    });

    return this.messageRepository.save(message);
  }

  async getMessagesBetweenUsers(user1Id: number, user2Id: number):
  Promise<Message[]> {
    return this.messageRepository.find({
      where: [
        { sender: { id: user1Id }, receiver: { id: user2Id } },
        { sender: { id: user2Id }, receiver: { id: user1Id } },
      ],
      relations: ['sender', 'receiver'],
      order: { timestamp: 'ASC' },
    });
  }

  async updateMessage(id: number, userId: number, content: string) {
    const message = await this.messageRepository.findOne({
      where: { id },
      relations: ['sender', 'receiver'],
    });
    if (!message) throw new NotFoundException('Message not found');

    if (message.sender.id !== userId) {
      throw new ForbiddenException('You can only edit your own messages');
    }
  }
}
```

```
message.content = content;
message.status = 'sent';
message.edited = true;
return await this.messageRepository.save(message);
}
```

```
async deleteMessage(id: number, userId: number) {
  const message = await this.messageRepository.findOne({
    where: { id },
    relations: ['sender', 'receiver'],
  });
  if (!message) throw new NotFoundException('Message not found');

  if (message.sender.id !== userId) {
    throw new ForbiddenException('You can only delete your own messages');
  }

  await this.messageRepository.remove(message);

  return message;
}
```

```
async sendMessageWithFile(senderId: number, receiverId: number, content: string,
fileUrl: string): Promise<Message> {
  const message = this.messageRepository.create({
    sender: { id: senderId } as User,
    receiver: { id: receiverId } as User,
    content,
    fileUrl,
    status: 'sent',
  });

  return this.messageRepository.save(message);
}
```

```
async getUserChats(userId: number) {
  console.log(`[Service]  getUserChats called with userId: ${userId}`);

  const messages = await this.messageRepository.find({
    where: [
      { sender: { id: userId } },
      { receiver: { id: userId } },
    ],
    relations: ['sender', 'receiver'],
  });
}
```

```

    order: { timestamp: 'DESC' },
  });

  console.log(`[Service] Found ${messages.length} messages`);

  const chatMap = new Map<number, {
    userId: number;
    name: string;
    lastMessage: string;
  }>();

  for (const msg of messages) {
    const otherUser = msg.sender.id === userId ? msg.receiver : msg.sender;

    if (!chatMap.has(otherUser.id)) {
      chatMap.set(otherUser.id, {
        userId: otherUser.id,
        name: otherUser.displayName,
        lastMessage: msg.content,
      });
    }
  }

  const result = Array.from(chatMap.values());
  console.log(`[Service] Returning ${result.length} chats`);
  return result;
}

async updateStatus(messageId: number, userId: number, status: 'delivered' | 'read') {
  const message = await this.messageRepository.findOne({
    where: { id: messageId },
    relations: ['receiver', 'sender'],
  });
  console.log(`[updateStatus] Incoming:`, { messageId, userId, status });

  if (!message) {
    console.warn(`[updateStatus] Message ${messageId} not found — possibly race condition`);
    throw new NotFoundException(`Message with id ${messageId} not found`);
  }

  if (message.receiver.id !== userId) {
    throw new ForbiddenException(`User ${userId} cannot update this message`);
  }
}

```

```

    message.status = status;
    await this.messageRepository.save(message);

    return message;
}

async create(senderId: number, receiverId: number, content: string) {
    const sender = await this.userService.findOneBy({ id: senderId });
    const receiver = await this.userService.findOneBy({ id: receiverId });

    if (!sender || !receiver) throw new NotFoundException('Sender or receiver not found');

    const message = this.messageRepository.create({
        sender,
        receiver,
        content,
        status: 'sent',
    });

    return await this.messageRepository.save(message);
}

async createMessage(
    senderId: number,
    receiverId: number,
    content: string = "",
    fileUrl?: string,
): Promise<Message> {
    const message = this.messageRepository.create({
        sender: { id: senderId } as User,
        receiver: { id: receiverId } as User,
        content,
        fileUrl,
        status: 'sent',
    });

    const savedMessage = await this.messageRepository.save(message);
    return this.messageRepository.findOneOrFail({
        where: { id: savedMessage.id },
        relations: ['sender', 'receiver'],
    });
}
}

```