МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА МЕСЕНДЖЕРА “ASHRISE MESSENGER” З ВИКОРИСТАННЯМ JAVASCRIPT, ELECTRON ТА MYSQL»**

Виконав: студент 4 курсу, групи 6.1212-пі-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Р.Є. Коваль
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Мильцев О.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп’ютерних наук, к.т.н. Добровольський Г.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ковалю Роману Євгеновичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка месенджера “AshRise Messenger” з використанням JavaScript, Electron та MySQL

керівник роботи Мильцев Олександр Михайлович, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Визначення вимог до інформаційної системи.
2. Проєктування інформаційної системи.
3. Реалізація та тестування інформаційної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	26.12.2025	
3.	Обробка методичних та теоретичних	30.01.2026	
	джерел.		
4.	Розробка першого та другого розділу.	20.03.2026	
5.	Розробка третього розділу.	14.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	15.06.2026	

Студент _____ Р.Є. Коваль
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.М. Мильцев
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка месенджера “AshRise Messenger” з використанням JavaScript, Electron та MySQL»: 55 с., 33 рис., 1 табл., 11 джерел.

БАЗА ДАНИХ, МЕСЕНДЖЕР, ШТУЧНИЙ ІНТЕЛЕКТ, AI ELECTRON, JAVASCRIPT, MYSQL, NODE.JS, PHOENIX.

Об’єкт дослідження – процес обміну миттєвими повідомленнями та медіаданими у кросплатформних мережевих середовищах.

Предмет дослідження – технології та інструментальні засоби розробки гібридних клієнт-серверних (веб та десктоп) застосунків.

Мета роботи – розробка безпечного кросплатформного месенджера з розширеною системою приватності та інтегрованим модулем штучного інтелекту.

Методи дослідження – методи об’єктно-орієнтованого програмування, проєктування архітектури клієнт-сервер, моделювання реляційних баз даних, асинхронного програмування та програмної інтеграції мікросервісів.

У роботі розроблено гібридний інформаційний застосунок (месенджер), який здатний функціонувати як у середовищі веббраузера, так і у вигляді повноцінної десктопної програми. Особливістю розробленої системи є впровадження суворої моделі приватності (система запитів у друзі та фільтрація небажаних повідомлень), використання гібридного зберігання даних MySQL на сервері, а також інтеграція нативного ШІ-помічника Phoenix AI для обробки текстових та голосових команд користувача. Результати роботи можуть бути використані для розгортання захищених корпоративних або приватних мереж комунікації, де критично важливим є високий ступінь контролю над особистими даними та відсутність надлишкового спам-трафіку.

SUMMARY

Bachelor's qualifying paper "Development of the Messenger "AshRise Messenger" Using JavaScript, Electron and MySQL": 55 pages, 33 figures, 1 table, 11 references.

AI, ARTIFICIAL INTELLIGENCE, DATABASE, ELECTRON, JAVASCRIPT, MESSENGER, MYSQL, NODE.JS, PHOENIX.

The object of the study is the process of instant messaging and media data exchange in cross-platform network environments.

The subject of the study is the technologies and tools for developing hybrid client-server (web and desktop) applications.

The aim of the study is the development of a secure cross-platform messenger with an advanced privacy system and an integrated artificial intelligence module.

The methods of research include object-oriented programming, client-server architecture design, relational database modeling, asynchronous programming, and software integration of microservices.

The paper presents the development of a hybrid informational application (messenger) capable of functioning both in a web browser environment and as a full-fledged desktop application. A special feature of the developed system is the implementation of a strict privacy model (a friend request system and unwanted message filtering), the use of MySQL data storage on the server, as well as the integration of a native AI assistant, Phoenix AI, for processing user text and voice commands. The results of the work can be used to deploy secure corporate or private communication networks where a high degree of control over personal data and the absence of excessive spam traffic are critically important.

ЗМІСТ

Завдання на кваліфікаційну роботу……..…..2
Реферат 4
Summary 5
Вступ..... 8
1 Визначення вимог до інформаційної системи 11
1.1 Постановка задачі та аналіз предметної області..... 11
1.2 Функціональні вимоги..... 13
1.3 Нефункціональні вимоги..... 15
2 Проєктування інформаційної системи..... 17
2.1 Проєктування бази даних..... 17
2.1.1 Концептуальна/ER діаграма 17
2.1.2 Логічне проєктування бази даних..... 19
2.2 Проєктування системи..... 20
2.2.1 Моделювання функціональних вимог через Use-Case діаграм 21
2.2.2 Моделювання бізнес-процесу за допомогою Activity Diagram (діаграми активностей)..... 25
2.2.3 Аналіз прецедентів через Sequence Diagram..... 28
2.2.4 Діаграма класів..... 30
2.2.5 Діаграма компонентів..... 33
2.2.6 Діаграма розгортання 35
3 Реалізація та тестування інформаційної системи 38
3.1 Особливості реалізації програми..... 38
3.1.1 Прелоад-скрипт як безпечний фундамент взаємодії..... 38
3.1.2 Гібридна архітектура: Desktop та Web 39
3.1.3 Оптимізація роботи з базою даних та мережею 40
3.1.4 Інтелектуальний модуль Phoenix AI 42

3.1.5 Нативні сповіщення та індивідуальні UX-рішення десктоп-версії 43
3.2 Демонстрація роботи інформаційної системи 44
3.2.1 Реєстрація та вхід до системи 44
3.2.2 Пошук користувачів та система дружби 46
3.2.3 Обмін повідомленнями та керування чатами 48
3.2.4 Створення та керування групами 52
Висновки 54
Перелік посилань 55

ВСТУП

Актуальність теми: стрімкий розвиток інформаційних технологій та цифровізація суспільства призвели до того, що системи обміну миттєвими повідомленнями (месенджери) стали невід'ємною частиною повсякденного, корпоративного та академічного життя. Сучасний користувач висуває високі вимоги до засобів комунікації, де ключовими критеріями є висока швидкість передачі даних, кросплатформеність, інтеграція з операційною системою, а також наявність інтелектуальних інструментів автоматизації.

Попри широкий вибір готових рішень на ринку, більшість з них є закритими комерційними продуктами, що обмежує можливості локального контролю над конфіденційними даними та гнучкої кастомізації під конкретні потреби користувачів чи організацій. Більше того, традиційні вебзастосунки часто мають обмежений доступ до системних ресурсів комп'ютера, що звужує можливості створення якісного користувацького досвіду (UX).

Вирішенням цієї проблеми є використання гібридного підходу до розробки програмного забезпечення. Фреймворк Electron дозволяє об'єднати гнучкість і швидкість вебтехнологій (HTML5, CSS3, JavaScript) з потужністю нативних десктопних застосунків. Створення месенджера "AshRise Messenger" на базі Electron у зв'язці з надійною системою керування базами даних MySQL та локальним кешуванням через SQLite дозволяє забезпечити стабільну роботу системи навіть за умов нестабільного мережевого з'єднання. Додаткова інтеграція персонального інтелектуального модуля "Phoenix AI" на базі мови Python відкриває нові можливості для взаємодії користувача із застосунком через текстові та голосові інтерфейси. Таким чином, розробка багатофункціонального кросплатформеного месенджера з нативною інтеграцією штучного інтелекту є актуальним науково-технічним завданням у сфері інженерії програмного забезпечення.

Мета дослідження – розробка кросплатформеної інформаційної системи обміну миттєвими повідомленнями "AshRise Messenger" з кастомною

архітектурою десктопного клієнта та інтегрованим інтелектуальним асистентом для підвищення ефективності та зручності користувацької комунікації.

Для досягнення поставленої мети необхідно вирішити такі **задачі дослідження**:

– проаналізувати сучасні підходи, технологічні стеки та наявні аналоги у сфері проєктування систем обміну повідомленнями;
– спроєктувати архітектуру месенджера, включаючи схему бази даних MySQL, локальний кеш SQLite та протоколи взаємодії між клієнтом і сервером;
– реалізувати десктопний клієнт на базі середовища Electron з використанням технології міжпроцесної взаємодії (IPC) для створення кастомних сповіщень;
– розробити та інтегрувати локальний модуль штучного інтелекту “Phoenix AI” для обробки текстових запитів та розпізнавання голосових повідомлень;
– провести комплексне тестування розробленого програмного забезпечення, оцінити продуктивність інтерфейсу та стабільність фонової синхронізації даних.

Об’єкт дослідження – процес обміну миттєвими повідомленнями та мультимедійними даними у кросплатформних мережевих інформаційних системах.

Предмет дослідження – методи, архітектурні рішення, шаблони проєктування та програмні засоби розробки клієнт-серверних десктопних застосунків на базі JavaScript, Electron, Node.js та MySQL з інтеграцією локальних моделей машинного навчання.

Методи дослідження: для вирішення поставлених задач було використано методи системного аналізу (для дослідження аналогів та формування вимог), об’єктно-орієнтованого проєктування та UML-моделювання (для побудови діаграм класів, компонентів, розгортання та

активностей), методи реляційного моделювання баз даних, а також методи інженерії програмного забезпечення для розробки, профілювання та тестування кросплатформеного софту.

Наукова новизна та практичне значення отриманих результатів: Практична цінність роботи полягає у створенні працездатного, оптимізованого гібридного месенджера, який поєднує ізольовану роботу процесів вебрендерингу та системні можливості ОС Windows. Розроблено унікальне UX-рішення у вигляді інтерактивних нативних сповіщень (notification.html), які працюють через IPC-канали без навантаження на основне вікно програми та дозволяють взаємодіяти з базою даних у реальному часі. Інтегрований ШІ-модуль “Phoenix AI” дозволяє реалізувати концепцію локального розумного помічника без необхідності використання дорогих хмарних API, що забезпечує високий рівень приватності. Створена система може бути використана як основа для захищених корпоративних месенджерів або приватних комунікаційних платформ.

Структура роботи: кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел.

1 ВИЗНАЧЕННЯ ВИМОГ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Постановка задачі та аналіз предметної області

Месенджери (від англ. *messenger* – кур'єр, посланець) – це спеціалізовані програмні додатки або вебсервіси, призначені для миттєвого обміну текстовими повідомленнями, медіафайлами, а також для здійснення голосових і відеодзвінків через мережу інтернет [1]. У сучасній інформаційній епосі вони стали основним інструментом як для повсякденного особистого спілкування, так і для побудови корпоративних комунікацій [2]. Типовими прикладами таких систем є світові лідери ринку, такі як Telegram, Viber, WhatsApp та Signal [3-6]. Їхня архітектура дозволяє користувачам швидко створювати особисті чати, об'єднуватися в групи, обмінюватися документами та використовувати системи push-сповіщень для оперативного реагування на нові події.

Незважаючи на високу популярність існуючих рішень, аналіз предметної області показує, що більшість сучасних месенджерів орієнтована виключно на масового споживача. Це часто призводить до перевантаження інтерфейсу зайвим нецільовим функціоналом (наприклад, стрічками новин, каналами, рекламою, вбудованими іграми), що відволікає користувача від основного завдання – комунікації [7]. Крім того, зростаючі вимоги до приватності та інтелектуальної обробки інформації створюють потребу в нових, більш адаптивних платформах, які б надавали користувачу повний контроль над своїм простором. Саме для вирішення цих проблем було розроблено інформаційну систему – AshRise Messenger.

Традиційні месенджери чудово справляються з базовою маршрутизацією повідомлень, проте вони мають низку обмежень. По-перше, популярні платформи часто не мають гнучких інструментів захисту від спаму на базовому рівні – користувачі змушені або приймати повідомлення від усіх незнайомців, або повністю закривати профіль [8]. По-друге, інтеграція

інструментів штучного інтелекту в них зазвичай реалізується через незручних сторонніх чат-ботів, які вимагають додаткових налаштувань. По-третє, багато систем є важкими монолітами, які працюють у закритих екосистемах [9].

На відміну від існуючих аналогів, розроблений AshRise Messenger пропонує унікальний баланс між мінімалізмом, безпекою та інноваційністю. До ключових переваг та можливостей системи, які відрізняють її від конкурентів, належать:

- *гібридна кросплатформна архітектура*: використання зв'язки Web-технологій (Express) та Electron дозволяє додатку працювати як у браузері, так і у вигляді повноцінної десктопної програми з локальним кешуванням бази даних для пришвидшення роботи [2, 10, 11];
- *глибока інтеграція ШІ*: наявність вбудованого інтелектуального модуля Phoenix AI, який функціонує як нативний помічник без необхідності підключення сторонніх плагінів;
- *персоналізація та організація*: наявність інтегрованого інструменту «Нотатки» (Notes), системи «Обраного» (Favorites) з можливістю сортування за типом контенту, а також гнучкі налаштування теми оформлення та мови інтерфейсу;
- *розширена система приватності*: застосунок має вбудовану систему «друзів» (Friend Requests) та унікальну функцію суворого обмеження комунікації, де писати особисті повідомлення можуть виключно підтверджені друзі, що повністю виключає спам.

Таким чином, головною задачею даної роботи є проєктування та реалізація програмного продукту, який перевершує класичні аналоги за рівнем приватності, має вбудовані інтелектуальні інструменти та забезпечує користувачу швидкий, безпечний і зручний обмін інформацією [1–11].

1.2 Функціональні вимоги

Функціональні вимоги визначають поведінку інформаційної системи та її реакцію на вхідні дані. Відповідно до поставлених завдань, розроблений програмний продукт «AshRise Messenger» повністю забезпечує цей вичерпний перелік операцій, необхідних для комплексного задоволення всіх потреб кінцевого користувача.

Управління обліковими записами та безпекою:

- реєстрація нових користувачів із обов’язковим підтвердженням електронної пошти через одноразові (OTP) коди;
- авторизація користувачів у системі з опціональною можливістю двофакторної перевірки при вході;
- відновлення доступу до акаунта (скидання пароля) та зміна існуючого пароля через підтвердження по email;
- управління особистим профілем: встановлення унікального тегу (наприклад, #user), нікнейму, інформації «Про себе», номера телефону та завантаження аватара.

Комунікація та обмін даними:

- створення особистих (P2P) чатів та миттєвий обмін текстовими повідомленнями;
- передача медіафайлів (зображень) у чатах;
- створення групових чатів із можливістю встановлення назви та аватара групи;
- управління груповими чатами: додавання нових учасників, вихід з групи та видалення учасників адміністратором (творцем) групи;
- редагування та видалення власних відправлених повідомлень;
- відповіді на конкретні повідомлення (reply) та автоматичне збереження невідправленого тексту у вигляді чернеток (drafts);
- закріплення (pin) важливих повідомлень у верхній частині екрана чату.

Соціальна взаємодія та приватність:

– гнучкі налаштування приватності: можливість обмежити отримання особистих повідомлень виключно від підтверджених «друзів»;
– пошук інших користувачів у системі за їхнім унікальним тегом;
– надсилання, прийняття або відхилення запитів на додавання до списку друзів;
– відстеження статусу активності користувачів у реальному часі (Online, Offline, Away).

Організація інформації та сповіщення:

– збереження важливих повідомлень у розділ «Обране» (Favorites) із можливістю пошуку та фільтрації за типом контенту (всі, текст, фото);
– захист обраних повідомлень (Protect): можливість заборонити іншим користувачам додавати ваші авторські повідомлення у свій список обраного;
– наявність системного ізольованого чату «Нотатки» для збереження особистої інформації користувача;
– система push-сповіщень та візуальних індикаторів (лічильників) для нових повідомлень, запитів у друзі та системних подій.

Інтеграція штучного інтелекту (Phoenix AI):

– взаємодія з вбудованим ІІ-асистентом через текстовий або голосовий ввід (на базі Python-мікросервісу);
– виконання системних та інформаційних команд (пошук у Google, відкриття YouTube, використання ChatGPT, встановлення таймерів тощо).

Налаштування інтерфейсу:

– зміна мови застосунку (українська, англійська, російська) із динамічним оновленням інтерфейсу без перезавантаження;
– зміна візуальної теми оформлення (світла/темна тема).

1.3 Нефункціональні вимоги

Нефункціональні вимоги визначають ключові критерії якості, системну архітектуру, а також обмеження й умови функціонування програмного продукту. Розроблена інформаційна система «AshRise Messenger» повністю відповідає цим критеріям, забезпечуючи високий рівень надійності, швидкодії та безпеки під час роботи.

Вимоги до архітектури та технологічного стека:

– клієнт-серверна архітектура: система повинна базуватися на гібридній архітектурі, що поєднує вебтехнології та десктопне середовище;
– серверна частина (Backend): повинна бути реалізована на базі платформи Node.js з використанням фреймворку Express для обробки HTTP-запитів та маршрутизації (API);
– клієнтська частина (Frontend/Desktop): десктопний застосунок має бути побудований за допомогою фреймворку Electron, що забезпечує кросплатформність (Windows) – вебверсія повинна коректно відображатися в усіх сучасних браузерах (Google Chrome, Safari, Mozilla Firefox);
– підсистема Штучного Інтелекту: модуль Phoenix AI має бути реалізований як окремий мікросервіс мовою Python 3, який взаємодіє з основним Node.js-сервером через породження дочірніх процесів (spawn).

Вимоги до зберігання даних: головна база даних: для централізованого зберігання облікових записів, історії повідомлень, списків друзів та налаштувань повинна використовуватися реляційна система управління базами даних MySQL.

Вимоги до безпеки та захисту інформації:

– хешування паролів: усі паролі користувачів повинні зберігатися в базі даних виключно у вигляді криптографічних хешів, згенерованих за допомогою алгоритму bcrypt із використанням солі (salt);

– захист доступу: система повинна підтримувати двофакторну верифікацію через електронну пошту за допомогою одноразових 6-значних кодів (OTP), термін дії яких обмежено (наприклад, 10 хвилин);
– ізоляція контексту: взаємодія між графічним інтерфейсом користувача (UI) та ядром системи в Electron-додатку повинна відбуватися виключно через безпечний міст (preload.js з використанням contextBridge та IPC-каналів), що унеможливлює пряме виконання шкідливого коду.

Вимоги до надійності та продуктивності:

– оновлення в реальному часі: для забезпечення миттєвої доставки повідомлень та сповіщень система повинна використовувати механізм Server-Sent Events (SSE) для вебклієнтів та IPC-трансляції для десктопних клієнтів;
– обробка медіафайлів: застосунок повинен підтримувати оптимізоване завантаження та зберігання зображень (BLOB-формат), використовуючи проміжну пам’ять (наприклад, multer memoryStorage) для швидкої маршрутизації файлів у БД;
– стійкість до відмов: у разі втрати з’єднання клієнтська частина повинна мати механізми коректної обробки помилок (відображення відповідних статусів, збереження локальних чернеток повідомлень).

Вимоги до ергономіки та зручності (Usability):

– інтерфейс має бути інтуїтивно зрозумілим, відповідати сучасним стандартам UI/UX-дизайну та підтримувати принципи мінімалізму;
– система повинна забезпечувати підтримку багатомовності (i18n) із можливістю зміни мови на льоту без перезавантаження сторінки;
– користувач повинен мати змогу адаптувати зовнішній вигляд системи під власні потреби (перемикання між світлою та темною темами).

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Проєктування бази даних

База даних інформаційної системи «AshRise Messenger» побудована виключно на основі реляційної системи управління базами даних (СУБД) MySQL. Цей вибір зумовлений необхідністю забезпечення надійної транзакційної цілісності, високої швидкості обробки запитів та гнучкості у побудові складних зв’язків між користувачами і повідомленнями у реальному часі.

Уся інформація централізовано зберігається на сервері, що гарантує 100% синхронізацію даних між різними пристроями користувача. Структура бази даних є нормалізованою, що мінімізує надлишковість даних та захищає їх від аномалій оновлення чи видалення. Основна логіка взаємодії між таблицями реалізована через систему первинних (Primary Key) та зовнішніх ключів, де базовими ідентифікаторами виступають унікальні ID користувачів, чатів та груп.

2.1.1 Концептуальна/ER діаграма

Концептуальна модель бази даних відображає архітектуру взаємозв’язків між ключовими сутностями месенджера. На рисунку 2.1 зображено концептуальну ER-діаграму бази даних.

Робота системи забезпечується взаємодією наступних основних логічних блоків, які нижче розглянемо.

Користувачі (users): головна сутність системи, яка містить облікові дані (email, хеш пароля), персоналізовану інформацію (нікнейм, тег, статус, аватар) та конфігурації профілю (тема, мова, налаштування приватності). З цією сутністю безпосередньо чи опосередковано пов’язані всі інші таблиці.

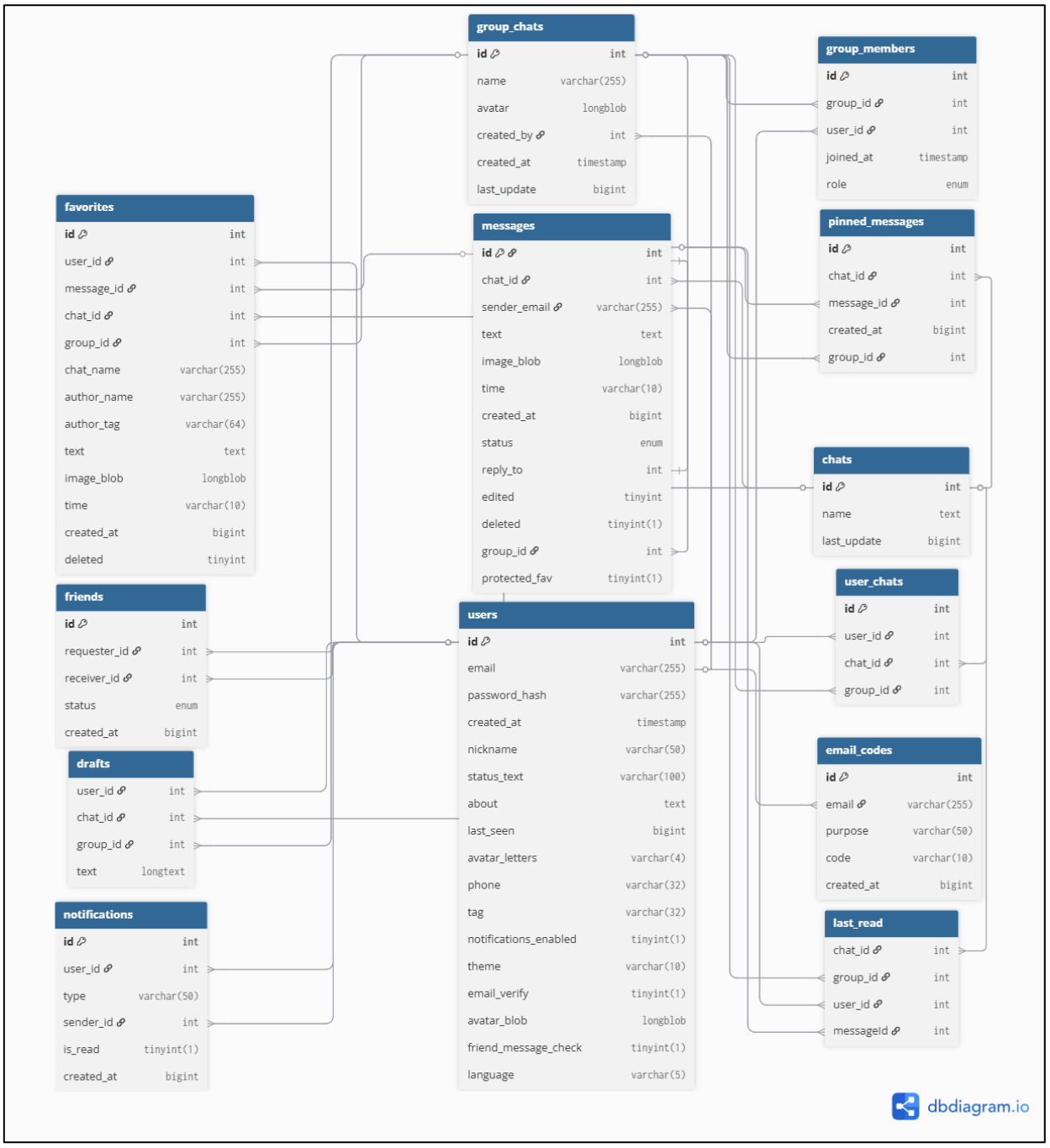


Рисунок 2.1 – ER-діаграма

Система комунікації (chats, group_chats, user_chats, group_members): система розділяє приватні діалоги (chats) та групові бесіди (group_chats). Приналежність користувачів до певного приватного чату визначається через зв'язуючу таблицю user_chats, а до груп – через таблицю group_members, де додатково фіксується роль (адміністратор або звичайний учасник).

Обмін даними (messages, drafts, pinned_messages): сутність messages є центральною для обміну інформацією та зберігає текст, медіафайли (BLOB),

статуси доставки, а також посилання на повідомлення, на яке дається відповідь (reply). Додатково реалізовано сутності для збереження невідправленого тексту (drafts) та закріплення важливих повідомлень у чаті (pinned_messages).

Соціальна взаємодія та організація (friends, favorites, notifications): сутність friends формує соціальний граф, фіксуючи запити у друзі між двома користувачами та їх статуси (очікування, прийнято, відхилено). Сутність favorites дозволяє користувачам зберігати важливі повідомлення в особистий простір, а notifications відповідає за фіксацію системних подій для інформування користувача.

Системні сутності (email_codes, last_read): відповідають за безпеку (зберігання OTP-кодів для верифікації та скидання пароля) та синхронізацію непрочитаних повідомлень (last_read).

2.1.2 Логічне проєктування бази даних

Логічне проєктування передбачає безпосередню реалізацію концептуальної моделі у вигляді конкретних таблиць із визначенням типів даних та ключових атрибутів. У таблиці 2.1 наведено повний перелік із 13 розроблених таблиць бази даних та їх призначення.

Таблиця 2.1 – Логічна структура бази даних MySQL

Назва таблиці	**Ключові атрибути**	**Призначення та опис сутності**
users	id (PK), email, password_hash, nickname, tag, avatar_blob, theme, language, email_verify, friend_message_check	Облікові записи користувачів, їхні налаштування інтерфейсу, приватності та персональні дані.
messages	id (PK), chat_id, group_id, sender_email, text, image_blob, status, reply_to, edited, deleted	Історія листування. Зберігає текстові повідомлення, зображення, статуси доставки та дані про редагування/видалення.
chats	id (PK), name, last_update	Реєстр приватних діалогів. Зберігає час останньої активності для сортування списку чатів.

Продовження таблиці 2.1

Назва таблиці	**Ключові атрибути**	**Призначення та опис сутності**
user_chats	id (PK), user_id, chat_id, group_id	Зв’язуюча таблиця, що визначає, до яких саме приватних чатів має доступ конкретний користувач.
group_chats	id (PK), name, avatar, created_by, last_update	Метадані групових чатів (назва, іконка групи, ідентифікатор творця групи).
group_members	id (PK), group_id, user_id, role	Перелік учасників кожної групи із зазначенням їхніх прав (admin або member).
friends	id (PK), requester_id, receiver_id, status	Соціальні зв’язки. Зберігає інформацію про відправлені заявки у друзі та їхній статус (pending, accepted, rejected).
favorites	id (PK), user_id, message_id, author_name, author_tag, text, image_blob	Збережені повідомлення («Обране»). Містить копії важливих повідомлень із зазначенням автора та чату.
pinned_messages	id (PK), chat_id, group_id, message_id	Реєстр закріплених повідомлень для відображення у верхній частині інтерфейсу діалогу.
drafts	user_id, chat_id, group_id, text	Збереження чернеток (невідправленого тексту), коли користувач перемикається між чатами.
notifications	id (PK), user_id, type, sender_id, is_read	Внутрішня система сповіщень про нові події (наприклад, заявки в друзі).
last_read	chat_id, group_id, user_id, messageId	Фіксація ідентифікатора останнього прочитаного повідомлення для коректної роботи лічильників.
email_codes	id (PK), email, purpose, code	Тимчасова таблиця для зберігання одноразових кодів верифікації та відновлення пароля.

2.2 Проєктування системи

На етапі проєктування програми здійснюється логічний перехід від сформульованих функціональних та нефункціональних вимог до безпосередньої побудови архітектури системи. Для всебічного, зрозумілого та стандартизованого опису інформаційної системи «AshRise Messenger»

використовується уніфікована мова моделювання (UML). Побудова відповідного набору діаграм дозволяє візуалізувати як статичну структуру коду, так і динамічну поведінку системи на різних рівнях абстракції.

У наступних підрозділах проведено поетапне моделювання застосунку через діаграми прецедентів (Use-Case), діаграми активностей (Activity), послідовностей (Sequence), класів (Class), компонентів (Component) та розгортання (Deployment). Такий комплексний підхід гарантує цілісність розробки, дозволяє виявити архітектурні недоліки ще до етапу написання коду та значно спрощує подальшу підтримку і масштабування програмного продукту.

2.2.1 Моделювання функціональних вимог через Use-Case діаграм

Діаграма прецедентів (Use-Case Diagram) є концептуальною моделлю, яка описує функціональне призначення інформаційної системи з точки зору взаємодії з її кінцевими користувачами (акторами). Вона відображає основні сценарії використання (прецеденти) та логічні зв’язки між ними. Для детального опису логіки месенджера розроблено дві основні діаграми, де єдиним ініціатором усіх процесів виступає актор «Користувач».

Перший модуль описує прецеденти екрана вітання: авторизації, реєстрації та базових налаштувань (рис. 2.2). Згідно з діаграмою, користувач має можливість самостійно взаємодіяти з інтерфейсними функціями, такими як «Зміна мови інтерфейсу» та «Зміна теми оформлення». Головні ж сценарії охоплюють процеси управління обліковим записом, які пов’язані між собою суворими залежностями. Наприклад, прецедент «Реєстрація акаунта» не є самостійним завершеним кроком – він обов’язково включає (відношення <<include>>) етап відправки коду на електронну пошту та подальше підтвердження реєстрації. Аналогічний жорсткий зв’язок наявний у прецеденті «Відновлення пароля», який обов’язково вимагає запиту коду для скидання та кроку встановлення нового пароля.

Гнучкішою є логіка прецеденту «Вхід в акаунт». Базовий сценарій обов'язково включає перевірку логіна та пароля. Проте, якщо в налаштуваннях профілю активовано додатковий захист (email_verify = 1), процес доповнюється через відношення розширення (<<extend>>): система вимагає запит коду підтвердження входу та його введення. Якщо ж опція вимкнена, цей крок ігнорується.

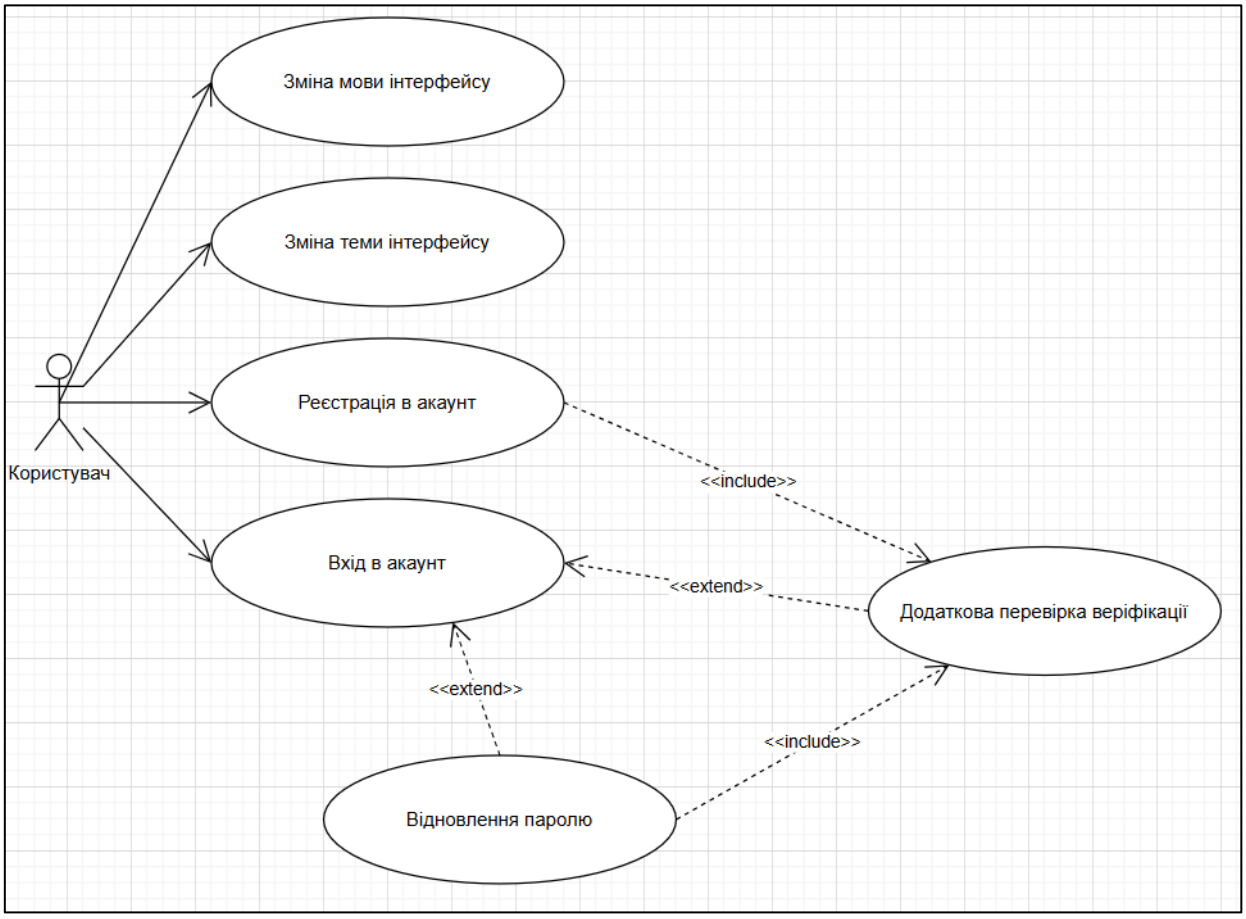


Рисунок 2.2 – Use-Case діаграма входу/реєстрації акаунту

Другий модуль (рис. 2.3) присвячений комплексній взаємодії користувачів з комунікаційним середовищем застосунку та глобальному управлінню чатами. Згідно з діаграмою, логіка спілкування розділяється на два магістральні сценарії: «Робота в приватному чаті» та «Робота в груповому чаті», які, у свою чергу, звертаються до спільного базового прецеденту «Робота в чаті». На цьому базовому рівні будь-який учасник діалогу (незалежно від його ролі чи типу чату) має змогу генерувати контент,

ініціюючи прецеденти «Написання та відправка повідомлення» та «Вибір фото та відправка його». Також усім учасникам доступний виклик окремої панелі для здійснення прецеденту «Пошук повідомлень», що дозволяє ефективно фільтрувати та знаходити необхідну інформацію в історії поточного листування.

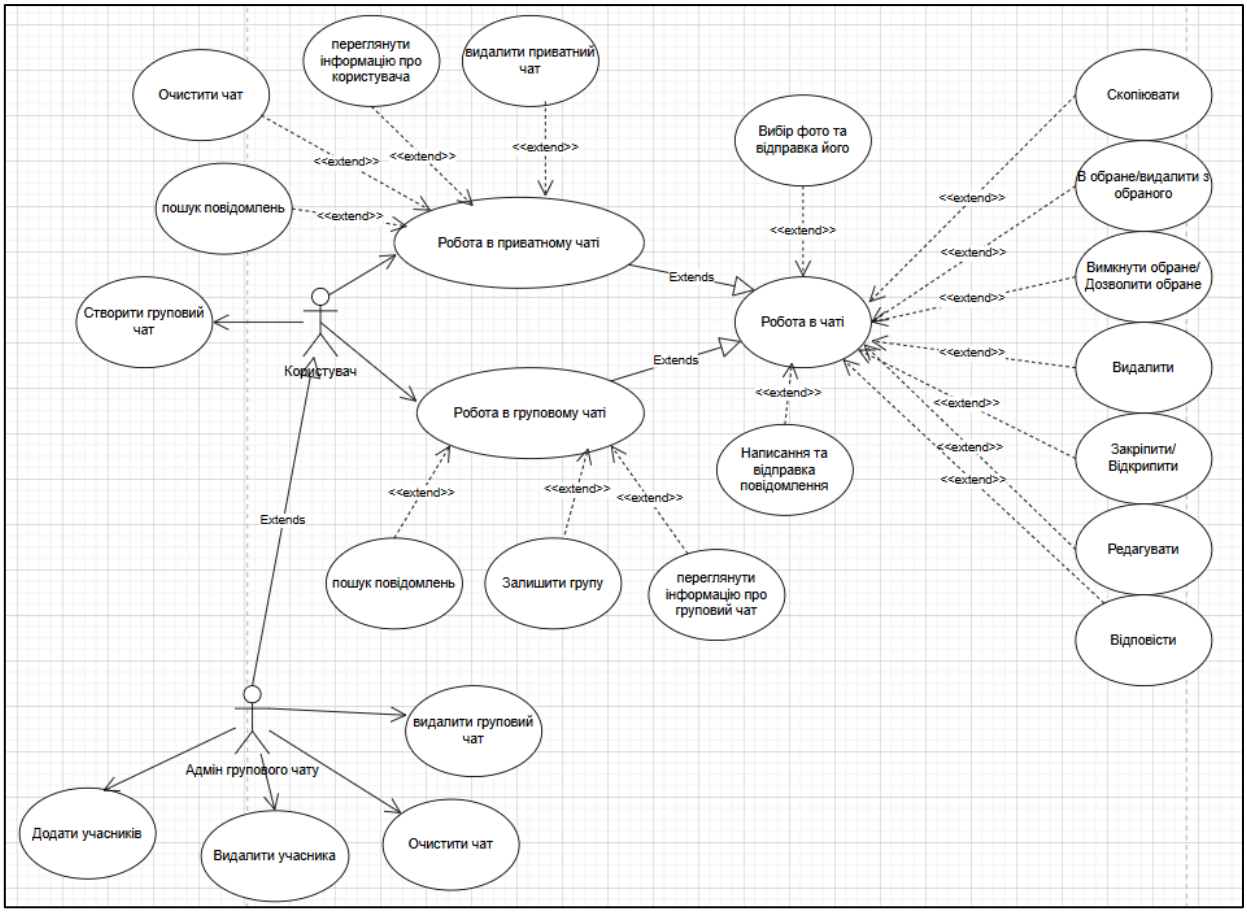


Рисунок 2.3 – Use-Case діаграма взаємодії користувачів в чаті

У контексті приватних (P2P) діалогів співрозмовники володіють абсолютно симетричними правами. Згідно зі схемою, користувач у приватному чаті може відкрити плашку «Інформація про співрозмовника» для перегляду профілю опонента. Крім того, він має повноваження ініціювати очищення історії листування («Очистити чат») або здійснити повне видалення діалогу («Видалити приватний чат») зі свого боку.

Важливою архітектурною особливістю цього модуля є ієрархічний розподіл ролей у групових бесідах. На схемі виділено спеціалізованого актора

– «Адмін групового чату», який успадковує всі властивості базового актора «Користувач», але наділяється розширеними управлінськими привілеями. Таким чином, хоча обидва актори можуть викликати панель «Інформація про групу», доступний їм функціонал розмежовується згідно з рівнем прав доступу, що забезпечує чітку структуру управління чатом.

Рядовий учасник групи через вікно інформації про групу може лише ознайомитися з її даними або ініціювати сценарій власного виходу («Залишити групу»).

Адміністратор (творець групи) через цю ж саму плашку «Інформація про групу» отримує доступ до інструментів модерації: він має право ініціювати прецеденти «Додати учасників» та примусово виключати користувачів через прецедент «Видалити учасника». Крім того, на рівні глобального управління чатом, лише адміністратору делеговано ексклюзивне право на радикальні дії: повне «Очистити чат» (для всіх) або «Видалити груповий чат» повністю.

Крім глобального управління самими діалогами, діаграма глибоко деталізує взаємодію на мікрорівні – безпосередню роботу з конкретними повідомленнями. Головною особливістю тут є розгалуження доступного функціоналу залежно від авторства обраного контенту, що реалізовано через точки розширення (виклик контекстного меню через клік правою кнопкою миші).

У випадку, коли користувач викликає контекстне меню на власному повідомленні, сценарій розширюється повним набором інструментів: користувач може відредагувати текст, безповоротно видалити повідомлення (для всіх учасників бесіди), відповісти на нього (Reply), скопіювати текст, закріпити (Pin) у верхній частині екрана або додати до особистого списку «Обране».

Натомість, у випадку взаємодії з чужим повідомленням, набір прецедентів є суворо обмеженим для захисту цілісності чужого контенту. Користувач не має доступу до редагування чи видалення, однак йому залишаються доступними функції копіювання, відповіді, закріплення та збереження повідомлення у своє «Обране».

Унікальним сценарієм на цій діаграмі є механізм управління авторським правом на контент. Якщо користувач зберіг чуже повідомлення у свій простір «Обране», оригінальний автор цього повідомлення має змогу ініціювати прецедент захисту.

Згідно з діаграмою, якщо автор натискає «Вимкнути обране» (захистити), це викликає сценарій розширення, який примусово ініціює скасування статусу обраного у всіх інших користувачів системи. Це гарантує творцю контенту повний контроль над його поширенням всередині месенджера.

2.2.2 Моделювання бізнес-процесу за допомогою Activity Diagram (діаграми активностей)

Для детального представлення динамічних аспектів системи та візуалізації бізнес-логіки було розроблено діаграму активностей (Activity Diagram). Дана діаграма описує ключовий бізнес-процес додатка – відправлення нового повідомлення в чат та подальше оновлення інтерфейсу користувача (див. рис. 2.4).

Для чіткого розмежування зон відповідальності процес розділений на три логічні доріжки (swimlanes), які наведено нижче.

Користувач – ініціатор процесу та кінцевий споживач результату.

Клієнтський застосунок / UI – фронтенд-частина системи, що відповідає за збір даних, взаємодію з сервером та відтворення інтерфейсу.

Сервер / Node.js Backend – серверна частина, що забезпечує валідацію даних, бізнес-логіку та взаємодію з базою даних.

Наведемо опис алгоритму виконання бізнес-процесу.

Ініціація: процес розпочинається з того, що користувач вводить текст повідомлення у відповідне поле та ініціює відправлення натисканням кнопки «Надіслати» або клавіші Enter.

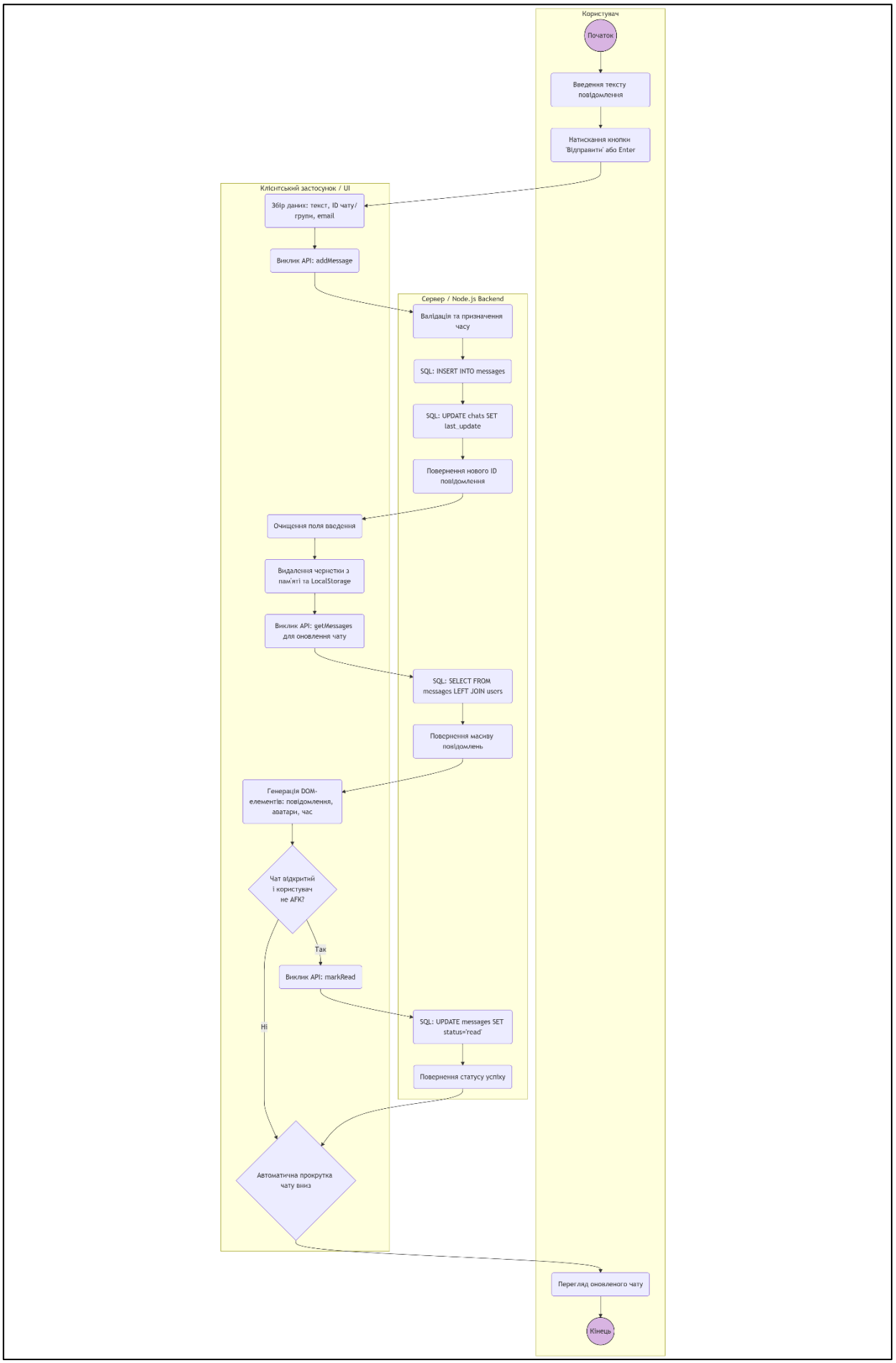


Рисунок 2.4 – Діаграма активностей

Формування запиту (UI): клієнтський застосунок перехоплює цю подію, збирає необхідні дані (текст повідомлення, ідентифікатор чату, належність до групи, email відправника) та здійснює виклик API-методу addMessage.

Обробка на сервері:

– сервер отримує запит, проводить валідацію вхідних даних та призначає точний час відправлення;
– виконуються транзакції у базі даних: додавання нового запису про повідомлення (SQL: INSERT INTO messages) та оновлення часу останньої активності у самому чаті (SQL: UPDATE chats SET last_update);
– після успішного збереження сервер повертає клієнту ідентифікатор (ID) нового повідомлення.

Очищення стану (UI): отримавши підтвердження від сервера, клієнтський застосунок очищує поле введення та видаляє збережений чернетку повідомлення з оперативної пам’яті та локального сховища браузера (LocalStorage).

Синхронізація даних: для відображення актуальної історії листування клієнт робить наступний виклик API – getMessages.

Вилучення даних (Backend): сервер виконує запит до бази даних (SQL: SELECT FROM messages LEFT JOIN chats), формує масив актуальних повідомлень і повертає його клієнтському застосунку.

Рендеринг інтерфейсу: на основі отриманого масиву UI-компонент динамічно генерує нові DOM-елементи: блок повідомлення, аватар користувача та час відправлення.

Перевірка статусу прочитання: далі у клієнтському застосунку спрацьовує умовний оператор, що перевіряє два фактори: чи відкрите в даний момент вікно чату і чи є користувач активним (не перебуває у статусі AFK).

– якщо умова істинна (так): викликається API-метод markRead. Сервер оновлює статус повідомлень у базі даних на «прочитано» (SQL: UPDATE messages SET status=read) та повертає статус успішного виконання;

– якщо умова хибна (ні): крок оновлення статусу прочитання пропускається.

Завершення циклу: незалежно від результату попередньої перевірки, клієнтський застосунок автоматично прокручує вікно чату вниз, щоб показати найсвіжіше повідомлення. Бізнес-процес завершується етапом, на якому користувач переглядає оновлений загальний чат.

Таким чином, побудована модель демонструє сувору послідовність дій, асинхронний характер взаємодії між клієнтом та сервером, а також коректну роботу зі станом інтерфейсу (очищення чернеток, автопрокрутка, статуси прочитання), що забезпечує високий рівень користувацького досвіду (UX).

2.2.3 Аналіз прецедентів через Sequence Diagram

Для дослідження часових аспектів взаємодії об’єктів у системі та деталізації обміну повідомленнями між різними рівнями архітектури було розроблено діаграму послідовності (Sequence Diagram). На відміну від діаграми активностей, цей тип моделювання дозволяє чітко візуалізувати життєвий цикл запитів та послідовність викликів методів між фронтендом, бекендом та базою даних.

На діаграмі представлено чотири основні учасники (lifelines):

– база даних (MySQL) – рівень збереження даних;
– клієнт (Renderer UI) – рівень представлення (React/фронтенд);
– користувач – зовнішній актор;
– сервер (API Bridge / Handlers) – рівень бізнес-логіки (Node.js).

Наведемо опис етапів взаємодії.

Надсилання повідомлення: процес ініціюється користувачем через введення тексту. Клієнтський застосунок надсилає асинхронний запит addMessage із корисним навантаженням (payload) до сервера. Сервер виконує послідовно дві операції в базі даних: створення запису про нове повідомлення

(INSERT) та оновлення мітки часу останньої активності в чаті (UPDATE). Після отримання підтвердження від БД, Сервер повертає клієнту ID створеного повідомлення.

Оновлення стану клієнта: після успішної відповіді сервера клієнт очищує інтерфейсні елементи (поле введення) та видаляє тимчасові дані з чернеток. Далі ініціюється автоматичний запит getMessages для синхронізації історії чату. Сервер виконує складний запит SELECT із використанням операції LEFT JOIN для отримання повідомлень разом із даними про користувачів. Отриманий масив даних передається на фронтенд для рендерингу DOM-елементів (тексту, аватара, мітки часу).

Логіка опрацювання статусів (фрагмент Alt): на діаграмі виділено комбінований фрагмент альтернативи (alt), який описує поведінку системи залежно від стану активності користувача:

– умова [вікно чату відкрите і користувач активний]: клієнт автоматично надсилає запит markRead; сервер оновлює статус повідомлень у БД на read – це забезпечує актуальність лічильників непрочитаних повідомлень у реальному часі;
– умова [користувач AFK або чат закритий]: система не виконує запит на оновлення статусу в БД, натомість на стороні клієнта відображається візуальний розділювач – лінія «нові повідомлення», що дозволяє користувачу пізніше ідентифікувати пропущений контент.

Фіналізація: на завершальному етапі клієнт виконує програмну прокрутку вікна чату вниз (scroll to bottom), забезпечуючи фокусування на останньому отриманому повідомленні, після чого користувач бачить фінальний результат оновлення.

Розроблена діаграма послідовності на рисунку 2.5 демонструє асинхронну модель взаємодії, яка мінімізує затримки інтерфейсу. Використання окремих етапів для збереження даних та подальшого їх отримання гарантує цілісність відображуваної інформації та дозволяє

реалізувати складну логіку відстеження статусів прочитання залежно від фокусу користувача.

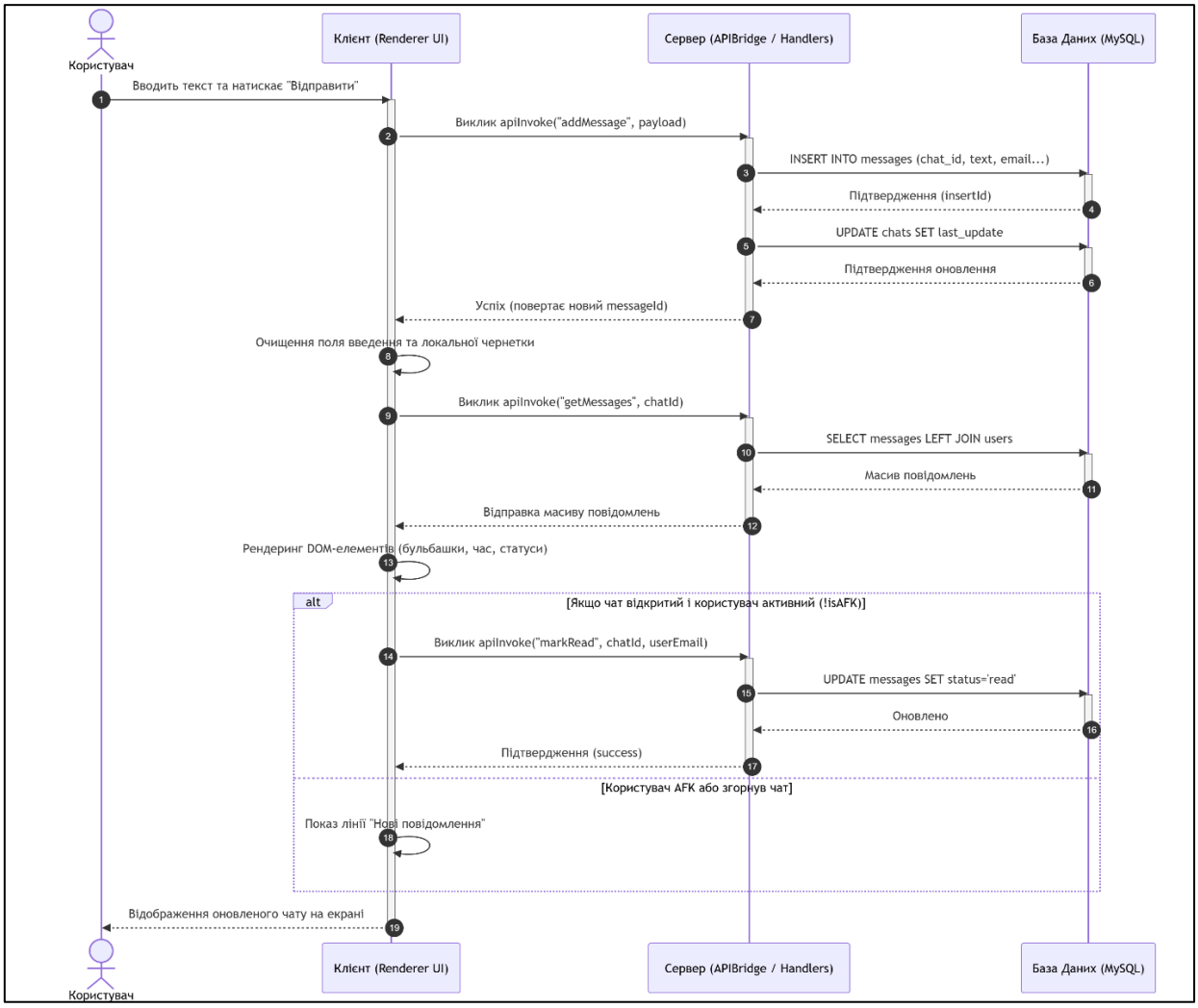


Рисунок 2.5 – Діаграма послідовності

2.2.4 Діаграма класів

Оскільки розроблювана система базується на модульній архітектурі JavaScript, де основною одиницею інкапсуляції є файл-модуль, для візуалізації статичної структури була використана діаграма класів (Class Diagram).

У даній моделі кожен клас репрезентує окремий логічний модуль системи, описуючи його методи та взаємозв’язки в процесі обробки та пересилання повідомлень.

Наведемо опис основних компонентів системи.

APIBridge (preload.js / IPC): проміжний шар, що забезпечує безпечну комунікацію між процесом рендерингу та основним процесом (Main Process). Використовує механізм apiInvoke та window.electronAPI для маршрутизації запитів до бекенд-обробників.

BackendHandlers (main.js): центральний вузол бізнес-логіки. Клас містить методи для обробки повідомлень (addMessage), отримання списків повідомлень (getMessages) та управління статусами прочитання (markRead).

Broadcaster (main.js): допоміжний модуль для розсилання повідомлень та подій між різними вікнами застосунку, включаючи ініціацію системних сповіщень.

Database (MySQL & SQLite): рівень персистенції даних. Модуль абстрагує роботу з конкретними СУБД, надаючи інтерфейси для виконання SQL-запитів (execute, run).

RendererUI (renderer.js): відповідає за рівень представлення. Містить методи для взаємодії з користувачем (sendMessage), відображення отриманих даних (renderMessages) та управління станом інтерфейсу. Також реалізує механізм фонового опитування сервера через setInterval.

NotificationUI (notification.js): окремий модуль інтерфейсу для взаємодії зі сповіщеннями, що дозволяє користувачу швидко відповісти або відмітити повідомлення як прочитане без відкриття основного вікна.

Розглянемо логіку взаємодії та потоки даних.

Потік відправлення: процес ініціюється в RendererUI, передається через APIBridge до BackendHandlers, де виконується запит INSERT до бази даних. Після успішного збереження тригериться подія через Broadcaster.

Потік синхронізації: завдяки методу setInterval у RendererUI відбувається регулярний запит нових даних. Це забезпечує актуальність чату в реальному часі без необхідності ручного оновлення.

Управління статусами: діаграма демонструє двосторонній зв’язок між інтерфейсом та базою даних для оновлення статусів повідомлень. Виклик

markRead може бути ініційований як з основного вікна чату, так і через швидкі кнопки в NotificationUI (метод markSingleMessageRead).

Представлена на рисунку 2.6 діаграма ілюструє чіткий розподіл обов'язків (Separation of Concerns). Використання APIBridge дозволяє ізолювати логіку роботи з даними від інтерфейсу користувача, що підвищує безпеку застосунку, а винесення логіки сповіщень у окремі модулі забезпечує масштабованість системи.

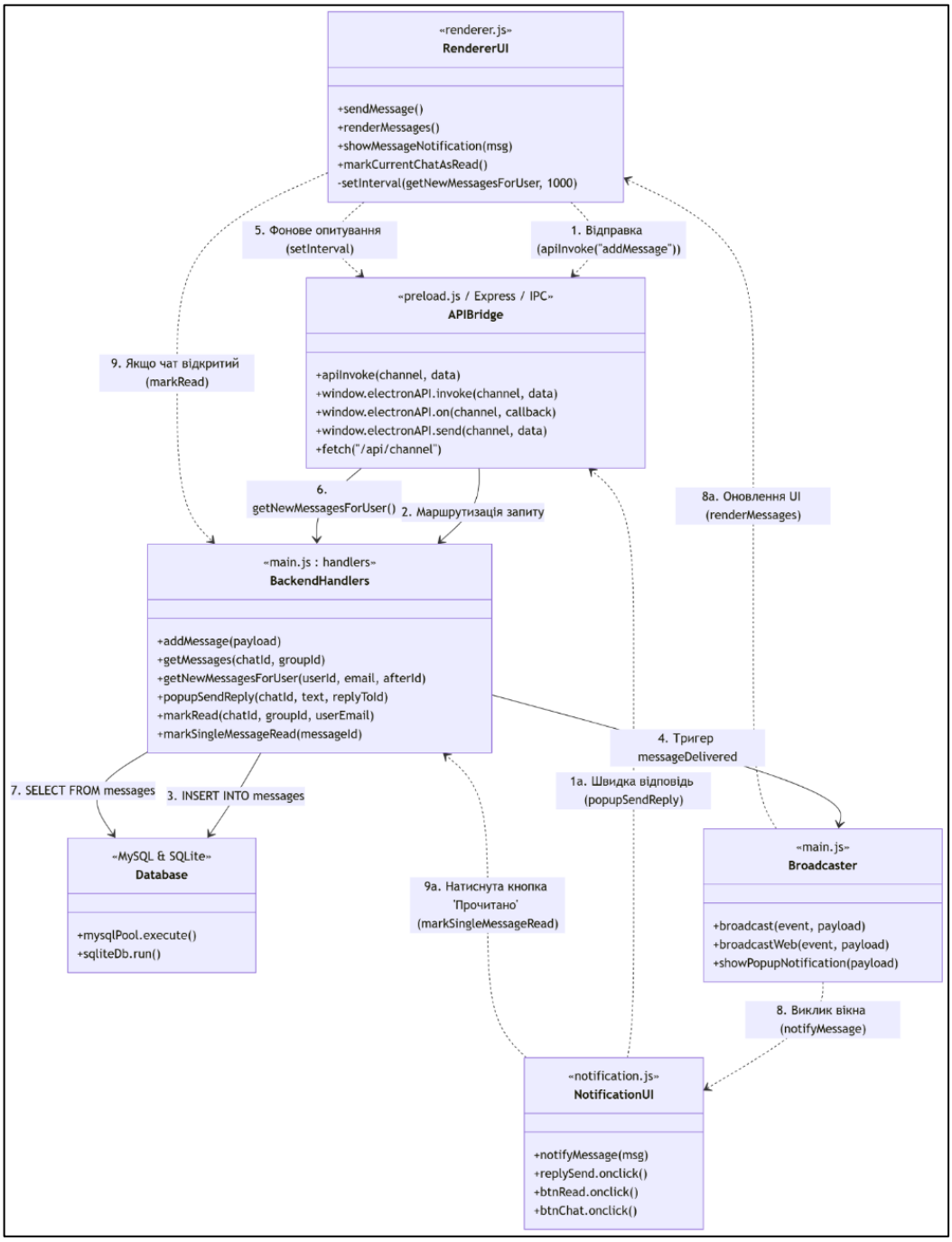


Рисунок 2.6 – Діаграма класів

2.2.5 Діаграма компонентів

Для візуалізації фізичної структури системи, її модульності та взаємозв'язків між програмними вузлами була розроблена діаграма компонентів (Component Diagram). Вона відображає розподіл функціональних можливостей між різними підсистемами та описує способи їхньої взаємодії (див. рис. 2.7).

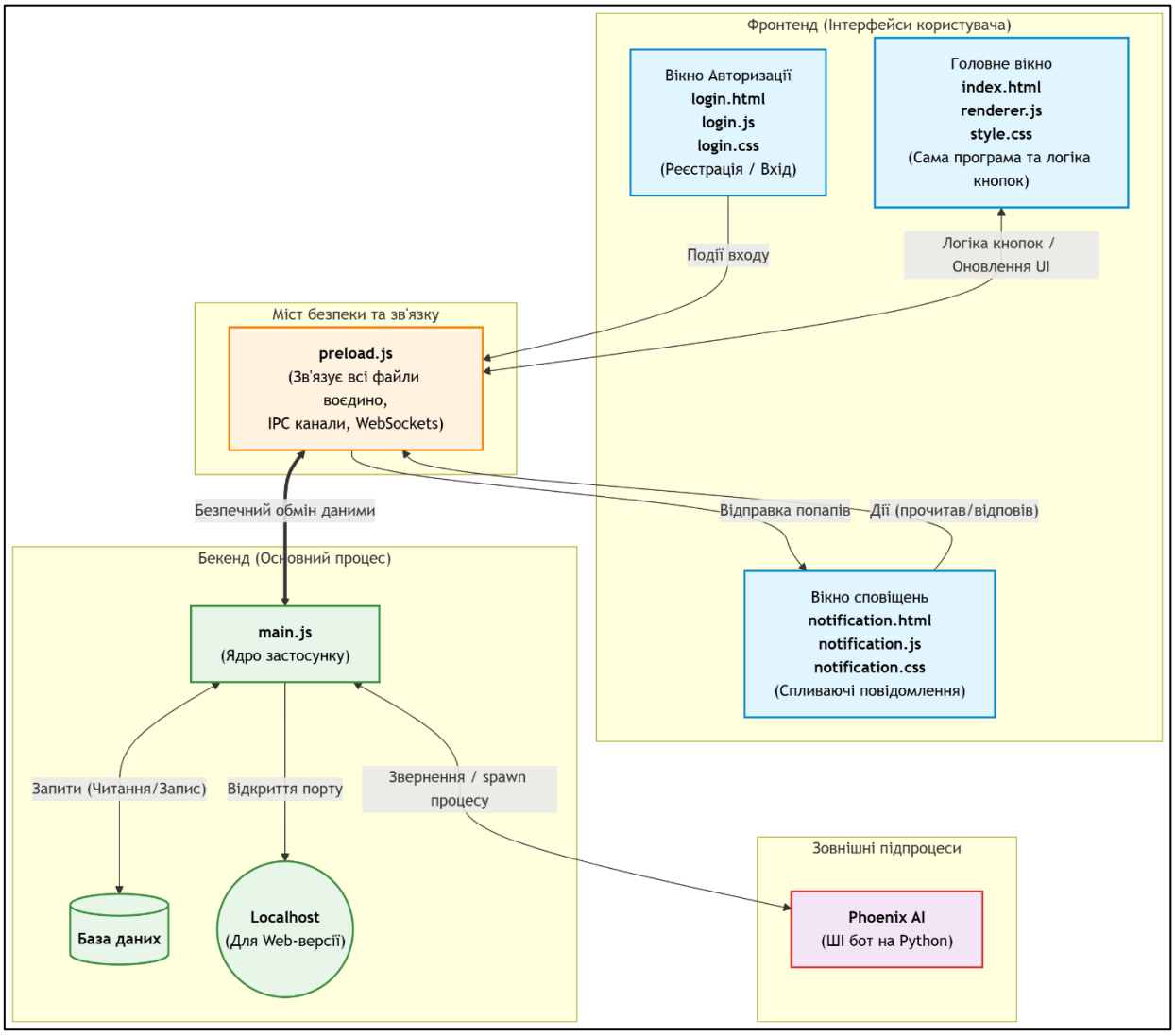


Рисунок 2.7 – Діаграма компонентів

Архітектура системи розділена на чотири ключові логічні блоки.

Фронтенд (інтерфейси користувача) – цей блок відповідає за візуальну взаємодію з користувачем і складається з трьох автономних модулів:

- вікно Авторизації (login.html, js, css): забезпечує вхід до системи та реєстрацію нових користувачів;
- головне вікно (index.html, renderer.js, style.css): є центральним компонентом інтерфейсу, де реалізована логіка кнопок, відображення чатів та управління контентом;
- вікно сповіщень (notification.html, js, css): окремий легковаговий модуль для відображення спливаючих повідомлень, що підтримує інтерактивні дії (швидка відповідь або відмітка про прочитання).

Міст безпеки та зв'язку (preload.js) – цей компонент виконує критичну роль посередника (Bridge) між процесами рендерингу (фронтендом) та основним процесом (бекендом). Він забезпечує:

- ізоляцію контексту для підвищення безпеки;
- зв'язування файлів через IPC-канали (Inter-Process Communication) та WebSockets;
- безпечний обмін даними, дозволяючи фронтенду викликати методи бекенда без прямого доступу до системних ресурсів.

Бекенд (Основний процес) – центральним компонентом є модуль main.js (Ядро застосунку). Його основними функціями є:

- управління даними: виконання запитів на читання та запис до Бази даних;
- мережева доступність: відкриття порту на Localhost для підтримки Web-версії застосунку;
- оркестрація процесів: ініціація відкриття нових вікон, відправка попапів та взаємодія із зовнішніми скриптами.

Зовнішні підпроцеси (Phoenix AI) – система підтримує модульну інтеграцію із зовнішніми інтелектуальними сервісами. Компонент Phoenix AI являє собою окремого ШІ-бота, написаного на мові Python. Взаємодія з ним відбувається через механізм звернення або spawn окремого процесу з ядра застосунку, що дозволяє винести ресурсомісткі обчислення за межі основного потоку Electron.

Взаємозв'язки та інтеграція: діаграма демонструє чітку спрямованість потоків даних. Фронтенд-модулі передають події через «Міст безпеки» до «Ядра», яке, у свою чергу, звертається до бази даних або зовнішнього ШІ-модуля. Зворотний зв'язок реалізований через механізм відправки попапів та оновлення UI, що забезпечує асинхронну та стабільну роботу застосунку.

Запропонована компонентна структура забезпечує високий рівень масштабованості та безпеки системи. Завдяки декомпозиції на окремі модулі (сповіщення, авторизація, AI-сервіс), систему легко модифікувати та доповнювати новими функціональними можливостями без порушення цілісності основного ядра.

2.2.6 Діаграма розгортання

Для опису фізичної топології системи, розподілу програмних компонентів по апаратних вузлах та визначення мережевих протоколів взаємодії було розроблено діаграму розгортання (Deployment Diagram). Дана модель дозволяє оцінити архітектурну стійкість застосунку та зрозуміти принципи його функціонування у реальному середовищі.

Розглянемо основні вузли (Nodes) системи.

Пристрій користувача (Desktop & Web):

– Desktop-версія: базується на середовищі Electron. вона розділена на два рівні: Renderer Process (відповідає за відображення інтерфейсу за допомогою технологій Chromium) та Main Process (виконує роль системного ядра на базі Node.js) – взаємодія між ними здійснюється через канали міжпроцесорної взаємодії (IPC канали);
– Web-версія: дозволяє доступ до системи через стандартні веббраузери (Chrome, Safari тощо) на смартфонах або ПК.

Сервер застосунку (Хостинг / Локальний сервер):

– центральним елементом є Node.js Runtime, на якому запущено Express Server (порт 3001);

– сервер виконує роль шлюзу, обробляючи запити від клієнтів через протоколи HTTP та SSE (Server-Sent Events) для забезпечення оновлень у реальному часі;
– до складу серверного вузла також входить підпроцес (Child Process), що забезпечує роботу Python 3 Runtime – саме тут розгорнуто модуль Phoenix AI (phoenix_api.py), який відповідає за інтелектуальну обробку даних.

Сервер баз даних: використовується MySQL Server (база даних ashrise_messenger). Взаємодія із сервером застосунку відбувається за специфічним MySQL Protocol через стандартний порт 3306.

Зовнішні сервіси (API): система інтегрована з Gmail SMTP Server для розсилки повідомлень або верифікації користувачів. Передача даних здійснюється через захищені протоколи SMTP / TLS (порти 465/587).

Особливості мережевої взаємодії:

– асинхронність: використання SSE (Server-Sent Events) дозволяє серверу миттєво надсилати повідомлення клієнтам без необхідності постійних запитів з їхнього боку, що критично для месенджера;
– гібридність: архітектура підтримує як локальне розгортання, так і хмарний хостинг, забезпечуючи доступність через десктопний клієнт та веббраузер;
– багатомовність середовища: поєднання Node.js для мережевої логіки та Python для ШІ-модулів дозволяє використовувати кращі інструменти для кожної специфічної задачі.

Діаграма розгортання, представлена на рисунку 2.8, підтверджує, що розроблена система має чітку триланкову архітектуру (Client-Server-Database) з додатковим рівнем для ШІ-сервісів.

Такий підхід гарантує високу доступність застосунку, безпеку даних за рахунок використання захищених протоколів та можливість легкого масштабування серверної частини незалежно від клієнтських терміналів.

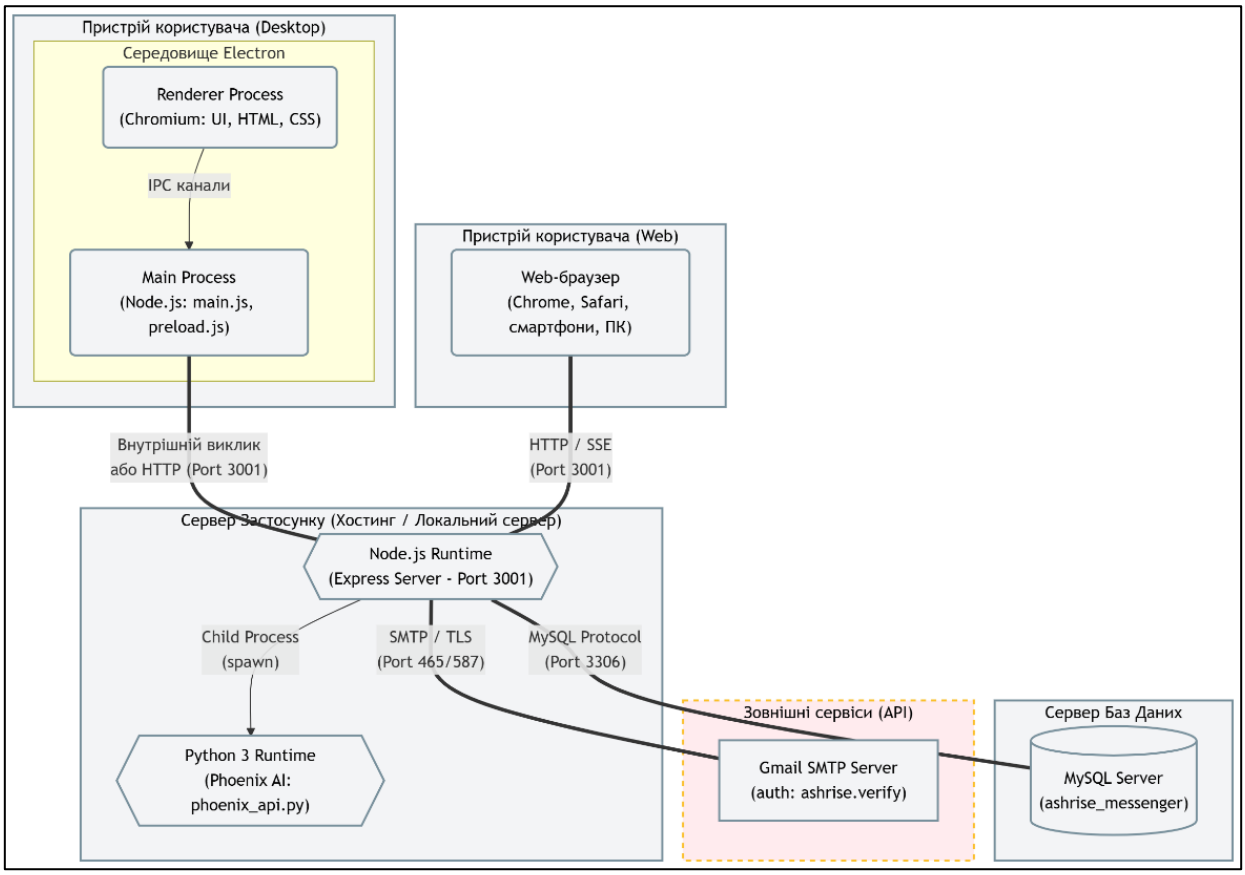


Рисунок 2.8 – Діаграма розгортання

Таким чином, розроблений комплекс UML-діаграм дозволив детально спроєктувати архітектуру, внутрішні процеси та структуру розгортання месенджера, створивши надійну теоретичну базу для його подальшої програмної реалізації.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Особливості реалізації програми

Програмна реалізація інформаційної системи базується на принципі модульності, де кожен файл виконує роль «цеглинки», що формує цілісну та стійку архітектуру «будинку» месенджера. Головною особливістю є синергія між різними середовищами виконання (Node.js, Python, Browser) та забезпечення безпечної взаємодії між ними.

3.1.1 Прелоад-скрипт як безпечний фундамент взаємодії

Центральною сполучною ланкою між графічним інтерфейсом та системними ресурсами є файл preload.js (див. рис. 3.1). Використовуючи технологію contextBridge, він створює безпечний ізольований «міст», який дозволяє фронтенду викликати потужні функції ядра Electron, не наражаючи систему на небезпеку прямих ін’єкцій коду.

```
const { contextBridge, ipcRenderer } = require("electron");
contextBridge.exposeInMainWorld("electronAPI", {
  invoke: (channel, data) => ipcRenderer.invoke(channel, data),
  on: (channel, callback) => ipcRenderer.on(channel, callback),
  send: (channel, data) => ipcRenderer.send(channel, data),
});
contextBridge.exposeInMainWorld("api", {
  getDraft: (data) => ipcRenderer.invoke("getDraft", data),
  saveDraft: (data) => ipcRenderer.invoke("saveDraft", data),
});
```

Рисунок 3.1 – Код файлу preload.js

3.1.2 Гібридна архітектура: Desktop та Web

Проєкт реалізований як універсальна платформа, що об’єднує дві принципово різні системи:

– Desktop (Electron): повнофункціональний застосунок із доступом до файлової системи, системних сповіщень та автозапуску – взаємодія відбувається через канали IPC (Inter-Process Communication);
– Web-версія: полегшена версія, доступна через браузер – система автоматично розпізнає середовище виконання: якщо window.electronAPI відсутній, застосунок перемикається на стандартні HTTP-запити через fetch до API-сервера.

Це дозволяє використовувати єдину логіку обробки повідомлень (renderer.js) для обох платформ, що значно спрощує підтримку та масштабування. Наприклад, на рисунках 3.2 та 3.3 продемонстровано одночасну роботу двох версій проєкту, а на рисунку 3.4 зображено процес їхнього запуску за допомогою однієї команди. Завдяки такому архітектурному рішенню вдалося мінімізувати дублювання коду, забезпечивши при цьому максимальну доступність месенджера для користувачів незалежно від обраного ними типу пристрою чи операційної системи.

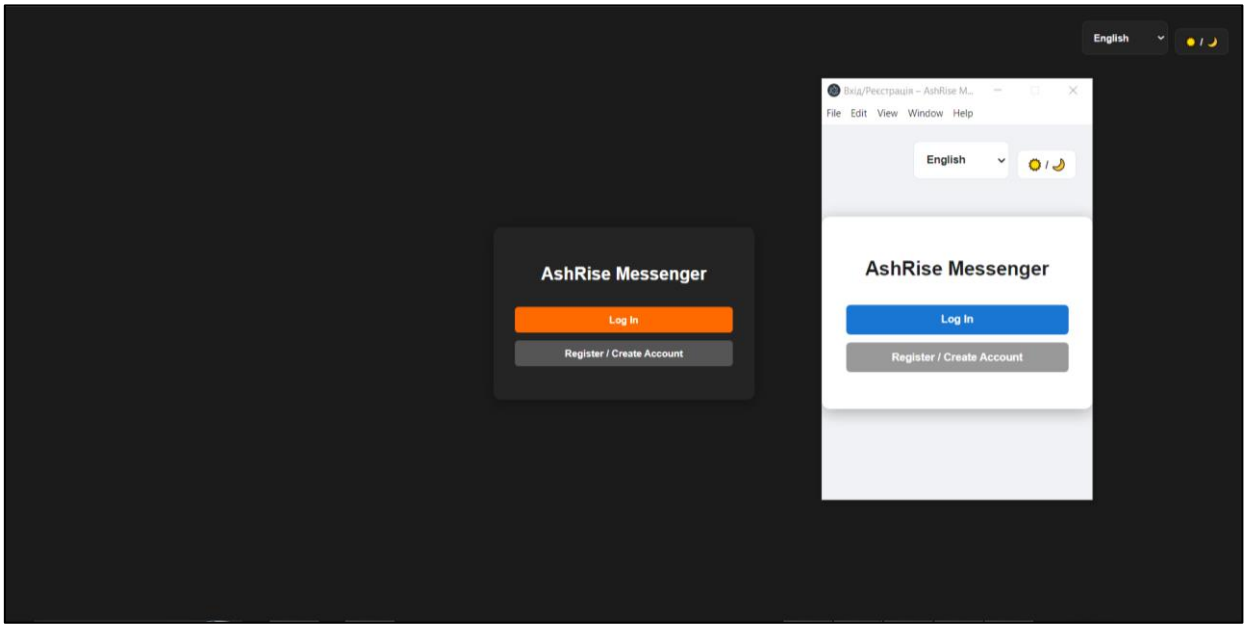


Рисунок 3.2 – Web та Dekstop версії логіну, місце реєстрації/входу

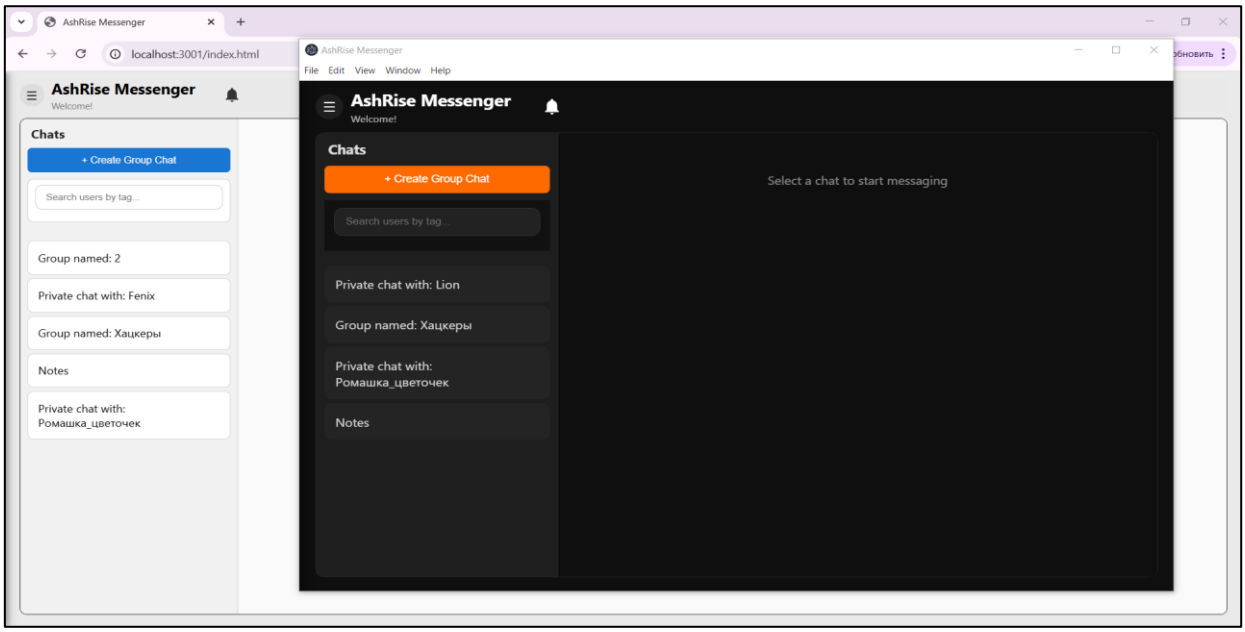


Рисунок 3.3 – Web та Dekstop версії проєкта, головний екран

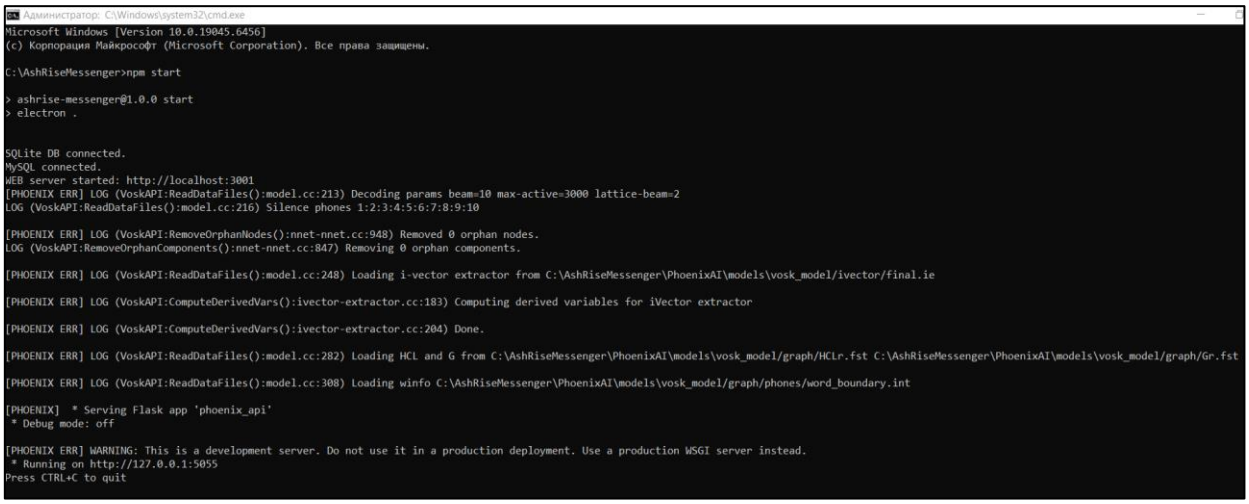


Рисунок 3.4 – Logs запуску проєкту

3.1.3 Оптимізація роботи з базою даних та мережею

Для досягнення максимальної швидкодії (відсутності «фрізів» та лагів) у проєкті реалізовано чіткий розподіл мережевих ресурсів.

Портова ізоляція: вебсервер Express працює на порту 3001, забезпечуючи зв’язок із клієнтами, тоді як база даних MySQL використовує порт 3306. Така архітектура дозволяє паралельно обробляти тисячі запитів без затримок у рендерингу інтерфейсу.

MySQL інтеграція: звернення до бази даних відбувається виключно через основний процес (main.js). Клієнтська частина через «прелоад» надсилає лише команду, а серверна частина виконує SQL-запит і повертає готовий результат.

Приклад коду, котрий з файлу main.js звертається до бази даних як для вебверсії наведено на рисунку 3.5, як для декстопної версії – рисунок 3.6.

```
async confirmRegister({ username, password, code, nickname, tag, language = 'ua' }) {
  const email = username.trim();
  const pass = password.trim();
  const userCode = code.trim();
  const [rows] = await mysqlPool.execute(
    `SELECT * FROM email_codes
     WHERE email=? AND purpose='register'
     ORDER BY created_at DESC LIMIT 1`,
    [email]
  );
  if (rows.length === 0) return { success: false, error: "Code not found" };
  if (rows[0].code !== userCode) return { success: false, error: "Invalid code" };
  if (Date.now() - rows[0].created_at > CODE_EXPIRE_MS)
    return { success: false, error: "The code has expired" };

  const hash = bcrypt.hashSync(pass, 10);
  const avatar_letters = nickname
    ? nickname.trim().split(" ").map(w => w[0]).join("").toUpperCase().slice(0, 2)
    : "??";
  await mysqlPool.execute(
    `INSERT INTO users
     (email, password_hash, nickname, status_text, about, last_seen, avatar_letters, phone, tag, language)
     VALUES (?, ?, ?, 'online', NULL, ?, ?, NULL, ?, ?)`,
    [email, hash, nickname, Date.now(), avatar_letters, tag, language]
  );
  const [[user]] = await mysqlPool.execute(
    "SELECT id FROM users WHERE email=?",
    [email]
  );
  const notesTranslations = {
    ua: "Нотатки",
    ru: "Заметки",
    en: "Notes"
  };
  const notesName = notesTranslations[language] || "Нотатки";
  const [chat] = await mysqlPool.execute(
    "INSERT INTO chats (name) VALUES (?)",
    [notesName]
  );
  const chatId = chat.insertId;
  await mysqlPool.execute(
    "INSERT INTO user_chats (user_id, chat_id) VALUES (?, ?)",
    [user.id, chatId]
  );
  await mysqlPool.execute("DELETE FROM email_codes WHERE email=?", [email]);
  return { success: true, userId: user.id };
},
```

Рисунок 3.5 – Звертання до бази даних через веб

```
ipcMain.handle("confirmRegister", (e, data) => handlers.confirmRegister(data));
```

Рисунок 3.6 – Звертання до бази даних через декстоп

3.1.4 Інтелектуальний модуль Phoenix AI

Особливою гордістю проєкту є інтеграція штучного інтелекту. Логіка ШІ винесена в окрему директорію і написана мовою Python. Це дозволяє:

– використовувати потужні бібліотеки обробки природної мови (NLP);
– запускати ШІ як окремий підпроцес (child process), що не навантажує основний цикл подій (Event Loop) месенджера – завдяки цьому користувач може спілкуватися з ботом у реальному часі, поки основний процес займається доставкою текстових повідомлень.

Логи, котрі показують запуск ШІ, можна побачити на рисунку 3.4 після запуску бази даних та Web server, а також на рисунку 3.7 та 3.8 показано як і сам репозиторій проєкту, так і індивідуальну папку ШІ.

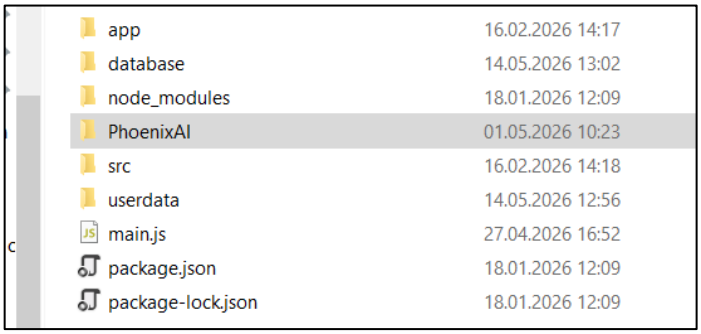


Рисунок 3.7 – Репозиторій проєкту

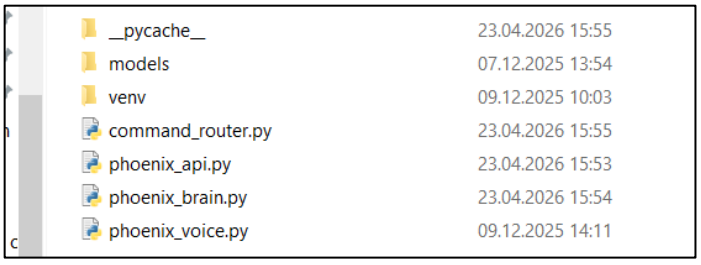


Рисунок 3.8 – Папка ШІ бота

3.1.5 Нативні сповіщення та індивідуальні UX-рішення десктоп-версії

Важливою особливістю реалізації є система кастомних інтерактивних сповіщень, яка є індивідуальною перевагою десктопної версії на базі Electron. На відміну від стандартної вебверсії, де можливості розробника обмежені стандартними та часто малофункціональними Push-сповіщеннями браузера, середовище Electron дозволило створити повністю контрольований елемент інтерфейсу.

Технічна унікальність: замість системних банерів, застосунок ініціює створення окремого легковагового вікна (notification.html), яке програмно позиціонується у правій частині робочого столу Windows. Завдяки параметрам alwaysOnTop: true та skipTaskbar: true, сповіщення з’являється поверх усіх активних вікон, але не захаращує панель завдань, імітуючи поведінку нативних елементів ОС.

Інтерактивна функціональність: це не просто статичне інформаційне вікно. Завдяки використанню ipcRenderer.send та ipcRenderer.invoke, користувач може взаємодіяти з повідомленням у реальному часі: надіслати швидку відповідь або позначити смс як прочитане. Ці дії миттєво передаються до бази даних MySQL, забезпечуючи фонову синхронізацію без необхідності розгортання основного вікна месенджера.

Оптимізація та продуктивність: оскільки сповіщення реалізоване як окремий процес рендерингу, воно не впливає на стабільність основної програми. Усі анімації появи та зникнення панелі оптимізовані за допомогою апаратного прискорення (GPU acceleration), що гарантує плавність інтерфейсу зі стабільною частотою кадрів навіть на апаратно слабких робочих станціях.

Такий підхід дозволив досягти глибинної інтеграції з операційною системою, забезпечуючи рівень UX (User Experience), який неможливо реалізувати в рамках стандартних вебтехнологій.

Вигляд повідомлення у декстоп версії наведено на рисунку 3.9.

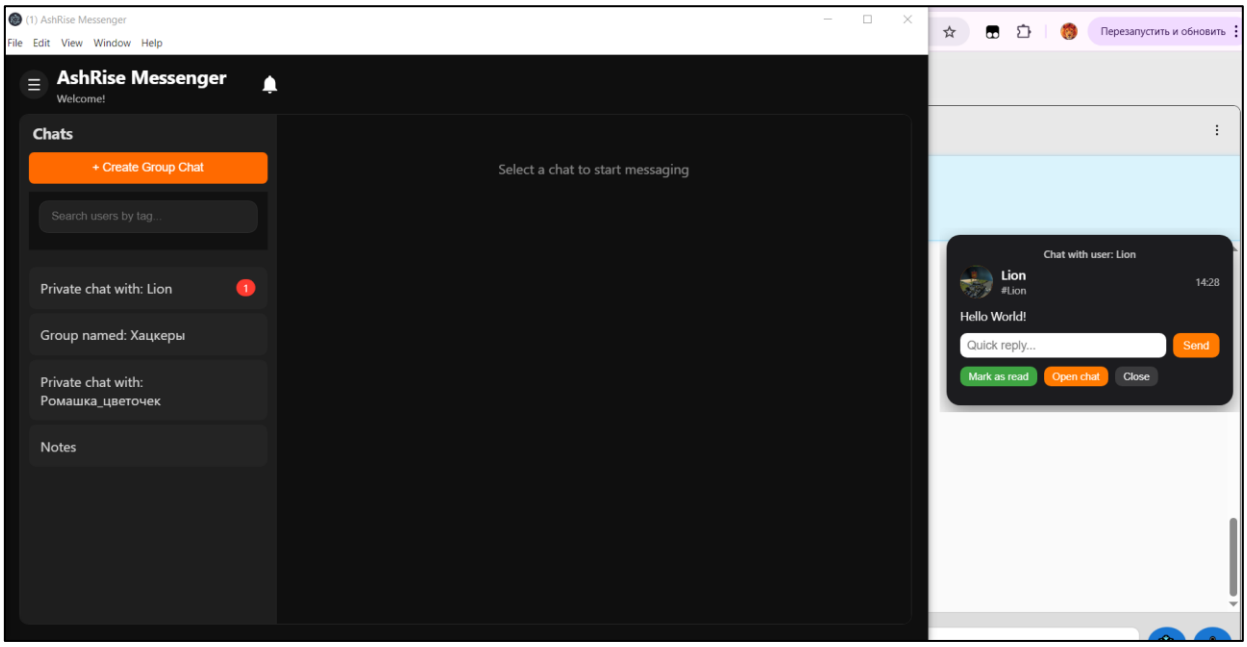


Рисунок 3.9 – Повідомлення від користувача

3.2 Демонстрація роботи інформаційної системи

У даному розділі описано практичну взаємодію користувача з інтерфейсом інформаційної системи та послідовність дій для використання основних функцій месенджера.

3.2.1 Реєстрація та вхід до системи

Процес початку роботи із застосунком максимально спрощений для кінцевого користувача:

Реєстрація: користувач обирає відповідний пункт у головному меню авторизації та заповнює реєстраційну форму, вказуючи Email, пароль, унікальний нікнейм та персональний тег (наприклад, #user). Після надсилання даних на вказану пошту надходить шестизначний код підтвердження. Тільки після введення цього коду акаунт стає активним і створюється автоматичний чат «Нотатки» для особистих записів (див. рис. 3.10).

Вхід: авторизація відбувається за Email та паролем. Залежно від налаштувань безпеки, система може вимагати додаткове підтвердження через пошту при кожному вході для захисту від несанкціонованого доступу (див. рис. 3.11).

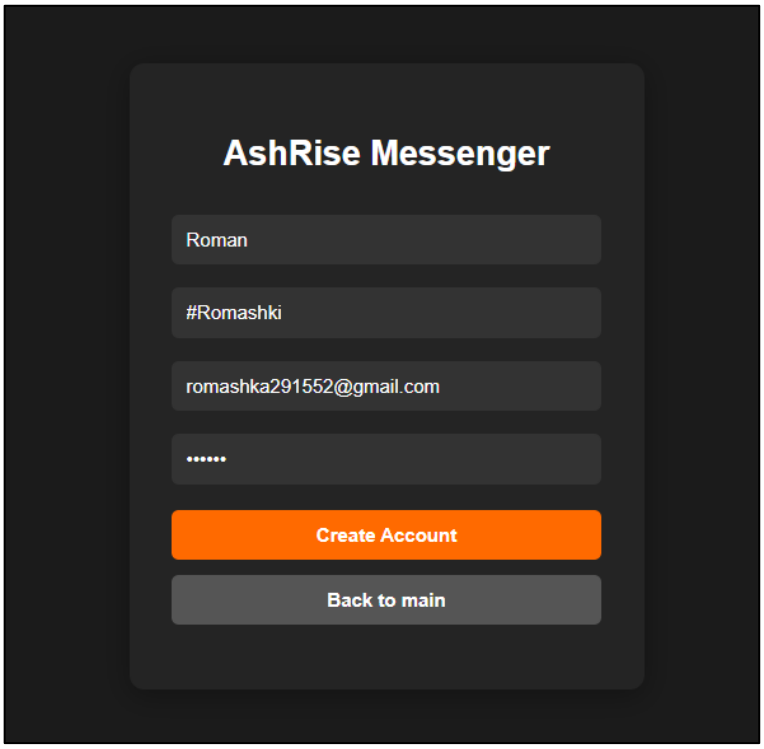


Рисунок 3.10 – Вікно реєстрації

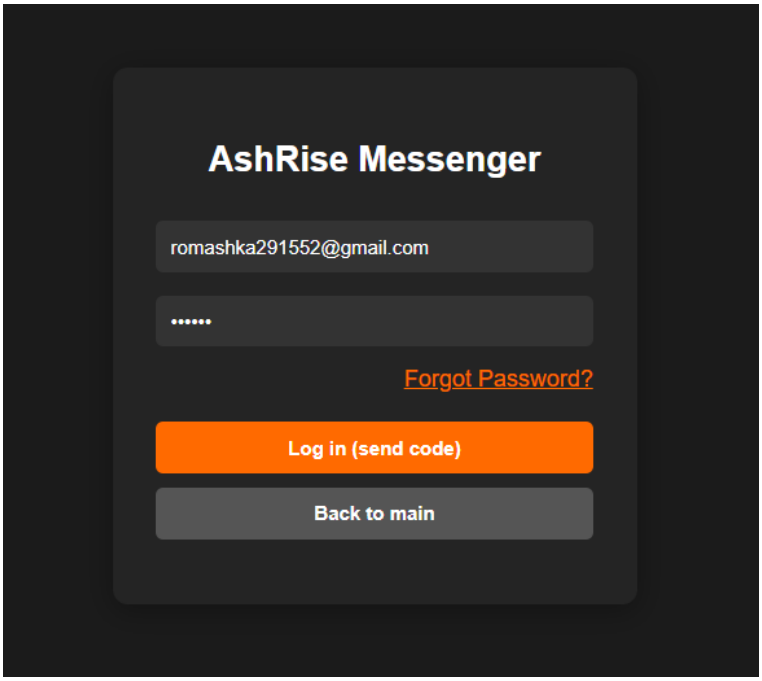


Рисунок 3.11 – Вікно входу

3.2.2 Пошук користувачів та система дружби

Для початку спілкування користувачу необхідно знайти співрозмовника.

Пошук: реалізовано через спеціальне поле в бічній панелі. Пошук здійснюється за персональним тегом. При введенні тегу (починаючи з символу #) система миттєво відображає картку знайденого користувача з його аватаром та нікнеймом (див. рис. 3.12 – 3.13).

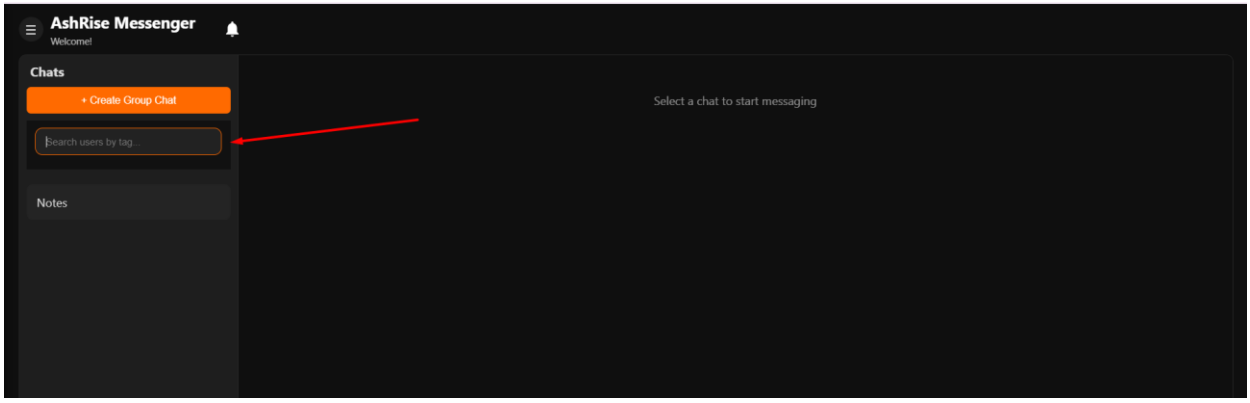


Рисунок 3.12 – Показана стрілкою пошукова система користувачів

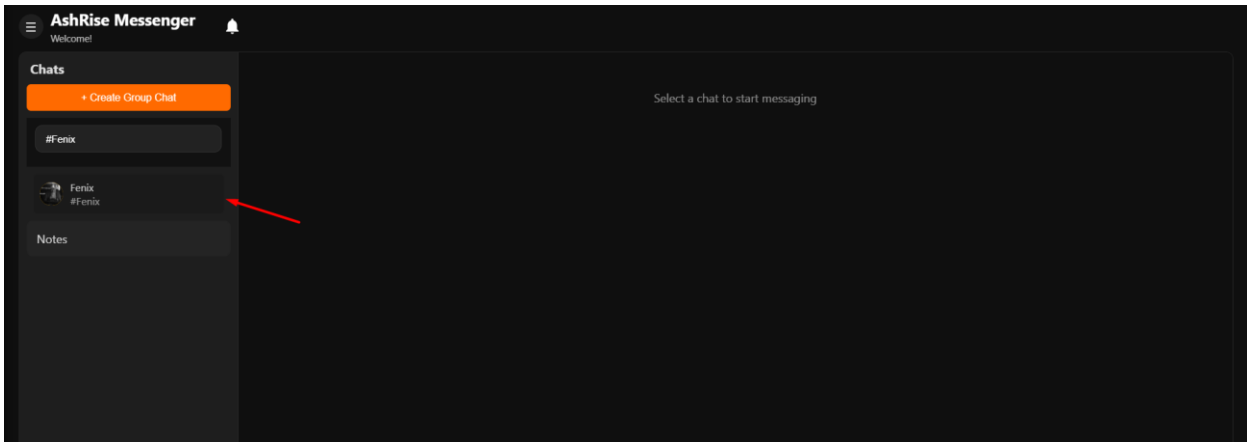


Рисунок 3.13 – Показано стрілкою знайденого користувача

Дружба: з картки знайденого користувача можна надіслати запит на додавання в друзі. Отримувач бачить сповіщення у вигляді червоного індикатора на іконці «дзвоника». У розділі «Друзі» користувач може прийняти або відхилити запит, а також переглянути список уже доданих друзів та їхній онлайн-статус (див. рис. 3.14 – 3.17).

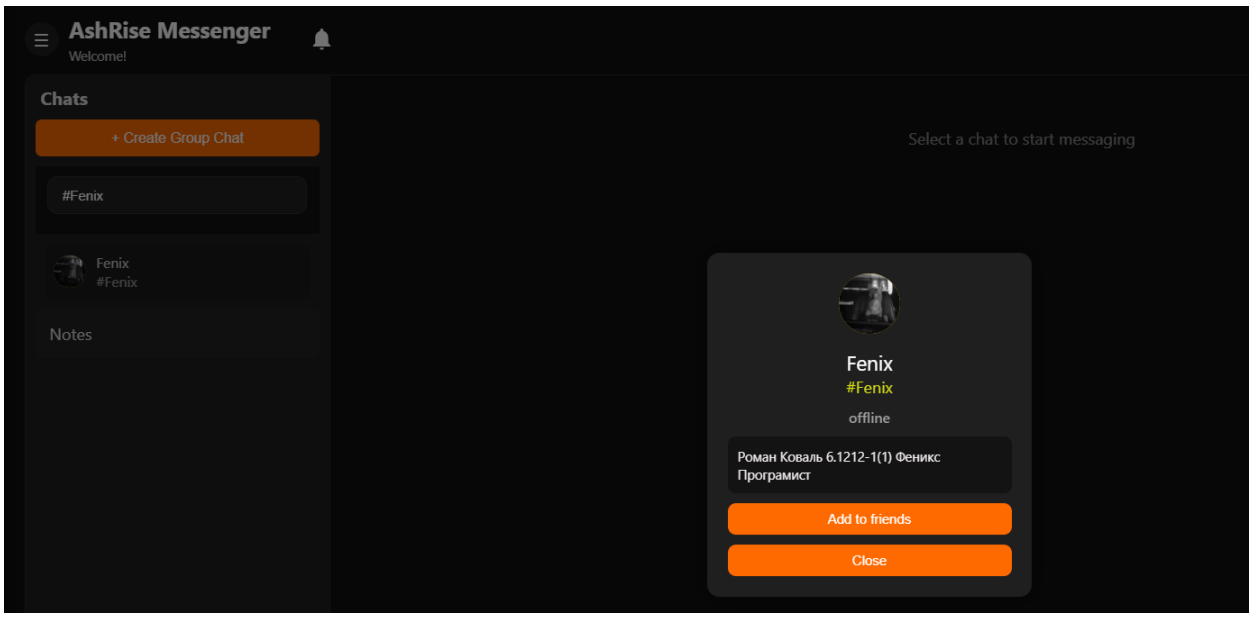


Рисунок 3.14 – Картка знайденого користувача

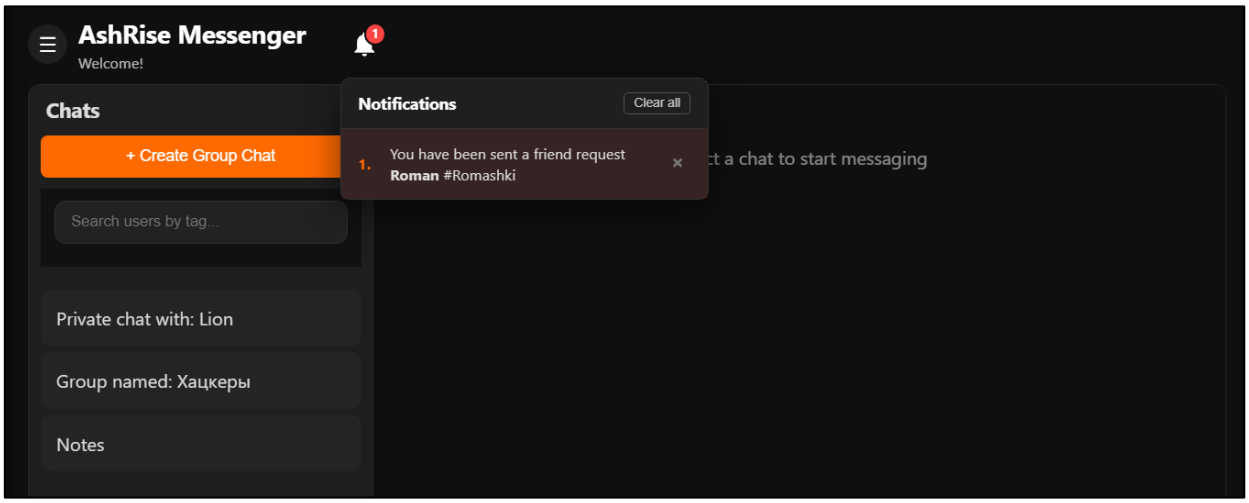


Рисунок 3.15 – Повідомлення у знайденого користувача,
що прийшла заявка у друзі

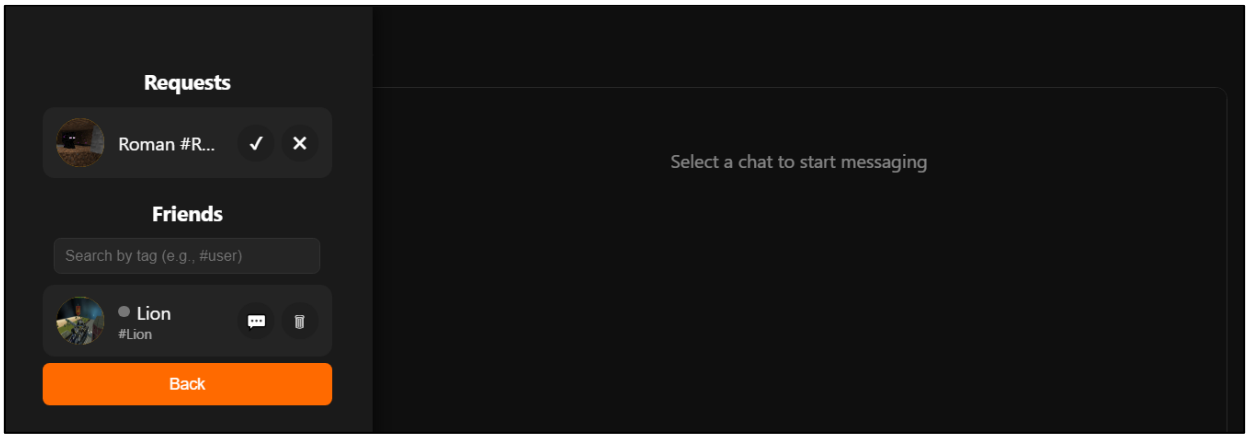


Рисунок 3.16 – Заявка у друзі у знайденого користувача

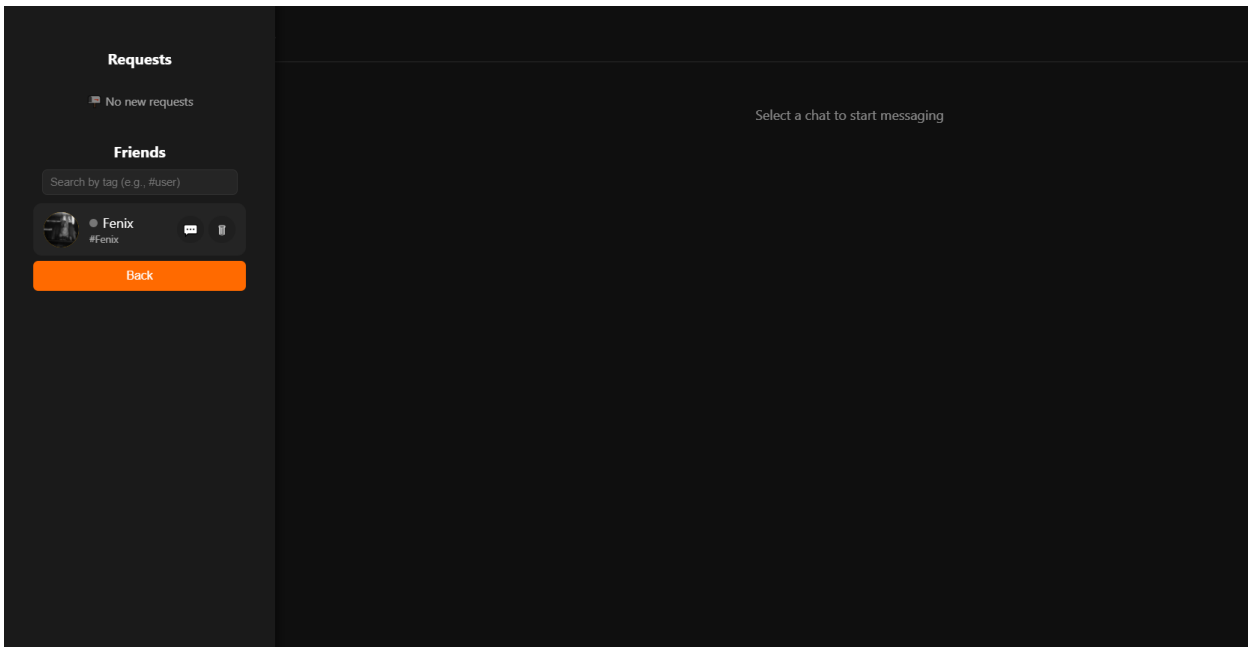


Рисунок 3.17 – Список друзів після підтвердження заявки знайденим користувачем

3.2.3 Обмін повідомленнями та керування чатами

Практична реалізація комунікаційного середовища месенджера базується на забезпеченні інтуїтивно зрозумілої взаємодії користувача з текстовим та мультимедійним контентом. Головні функціональні можливості цього модуля, черговість їхнього виконання та відповідні графічні інтерфейси розглядаються нижче.

Ініціація та створення приватних чатів. Користувач має можливість оперативно розпочати новий діалог безпосередньо через картку профілю іншого учасника або обравши його зі списку друзів (див. рис. 3.18). Система автоматично перевіряє наявність наявного діалогу в базі даних і, у разі його відсутності, динамічно створює новий сеанс зв'язку. Надіслані повідомлення підтримують інтерактивну систему статусів доставки для візуального контролю (одна галочка – надіслано на сервер, дві галочки – доставлено на пристрій отримувача, колірна індикація – повідомлення прочитано), що продемонстровано на рисунку 3.19.

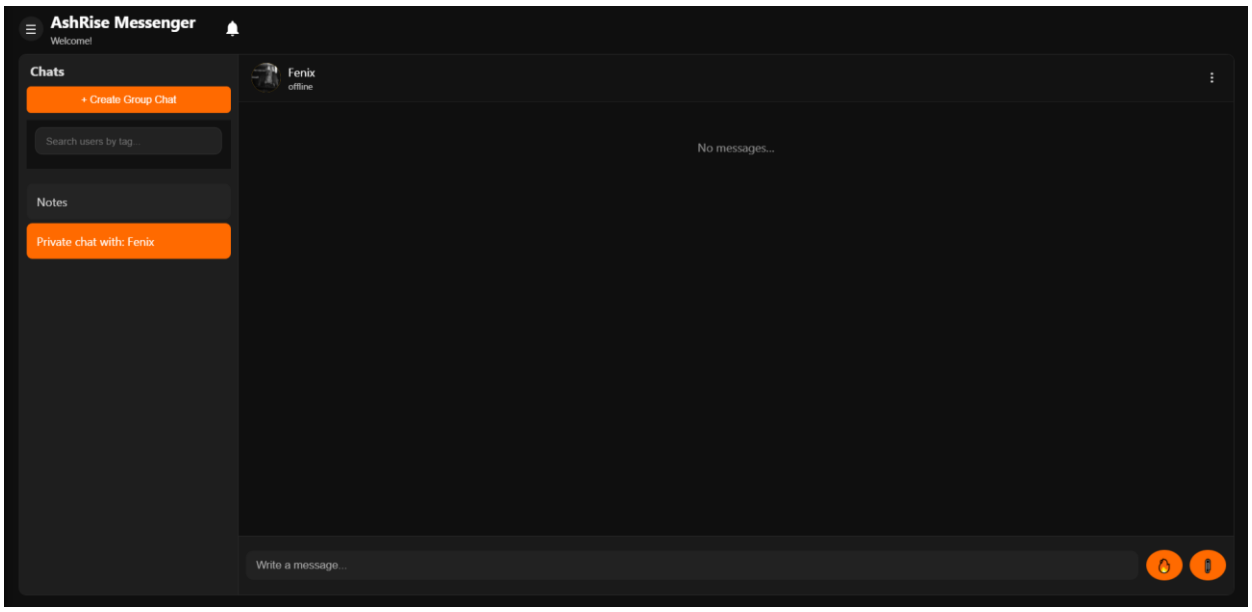


Рисунок 3.18 – Відкриття (створення) чату з користувачем

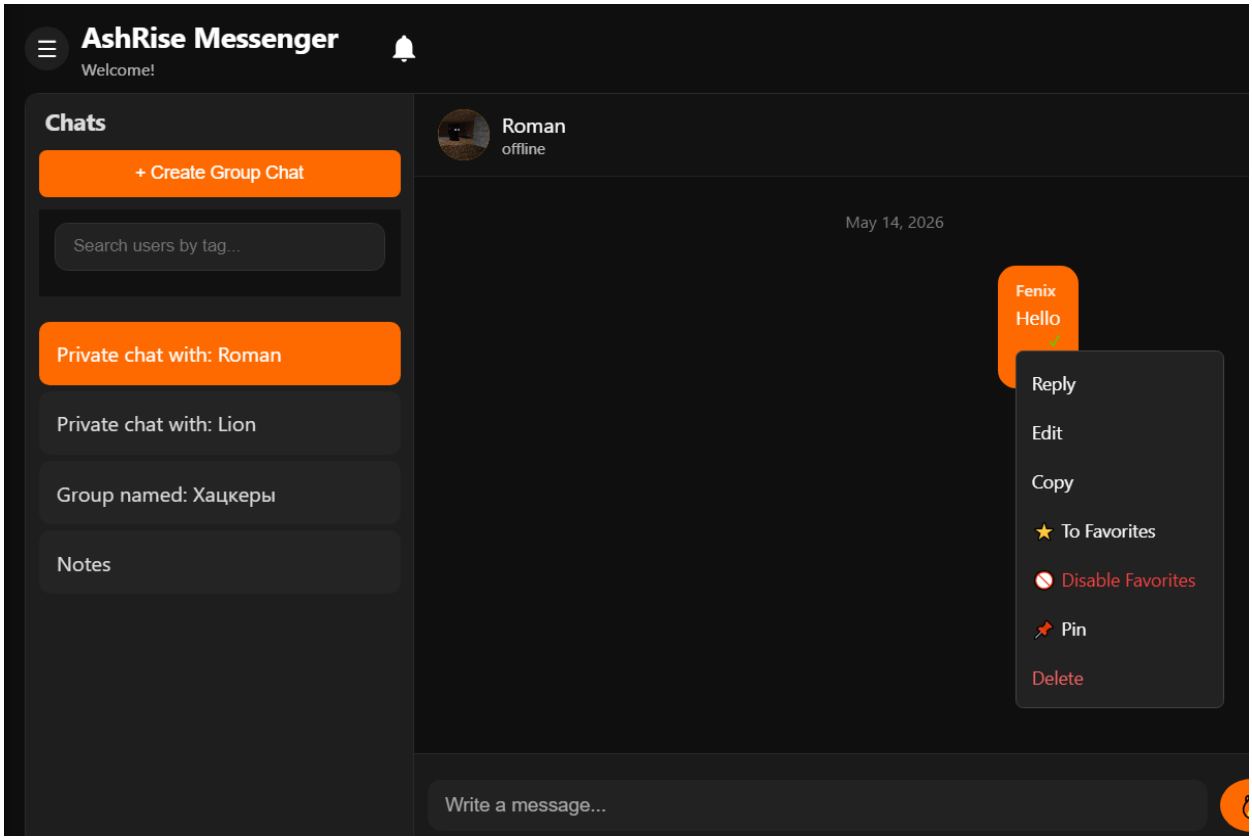


Рисунок 3.19 – Контекстне меню зі своїм повідомленням

Контекстні дії з повідомленнями. Для мікроуправління контентом у вікні чату реалізовано механізм виклику контекстного меню за допомогою натискання правої кнопки миші на повідомленні. Набір доступних опцій гнучко адаптується залежно від авторства обраного об'єкта. При взаємодії з

власним повідомленням користувачеві доступний повний спектр дій: редагування тексту, безповоротно видалення, копіювання, відповідь (Reply), закріплення у шапці чату або збереження в «Обране» (див. рис. 3.20). При кліку на чуже повідомлення функції редагування та видалення блокуються з метою захисту цілісності даних.

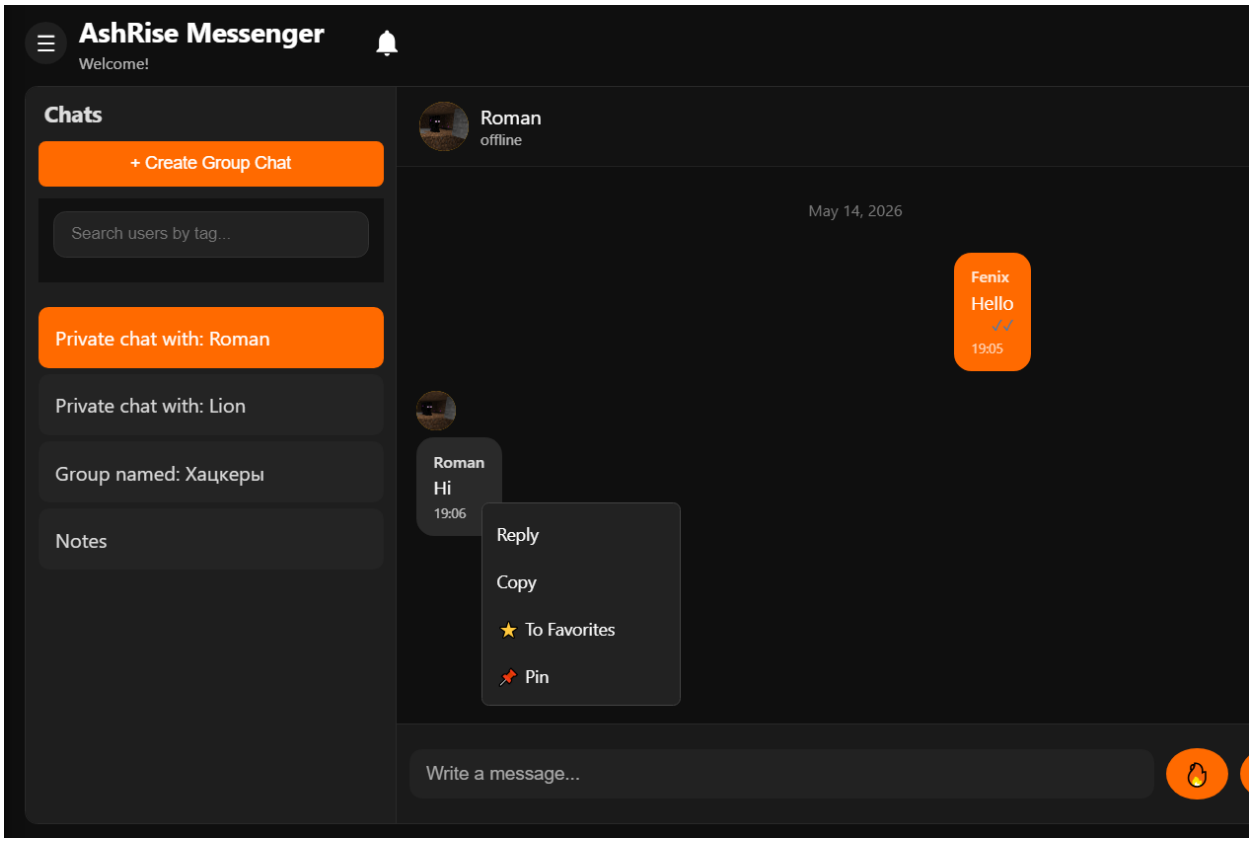


Рисунок 3.20 – Контекстне меню з чужим повідомленням

Глобальне керування діалогами. Окрім роботи з окремими репліками, у застосунку передбачено окреме меню для адміністрування чату в цілому (див. рис. 3.21). Через цей інтерфейс кожен користувач може скористатися функціями локального пошуку за ключовими словами в історії листування, тимчасового очищення вмісту екрана, повного видалення приватного чату зі свого боку або швидкого переходу до картки профілю співрозмовника.

Обмін мультимедійними даними. Програма підтримує передачу графічних файлів у реальному часі. Процес взаємодії побудований через інтеграцію із системним провідником: користувач ініціює вибір файлу через

діалогове вікно операційної системи, після чого зображення передається у клієнтську частину для попереднього перегляду та підтвердження відправки (див. рис. 3.22). У вікні листування надіслані фотографії автоматично масштабуються та відображаються у вигляді зручних мініатюр.

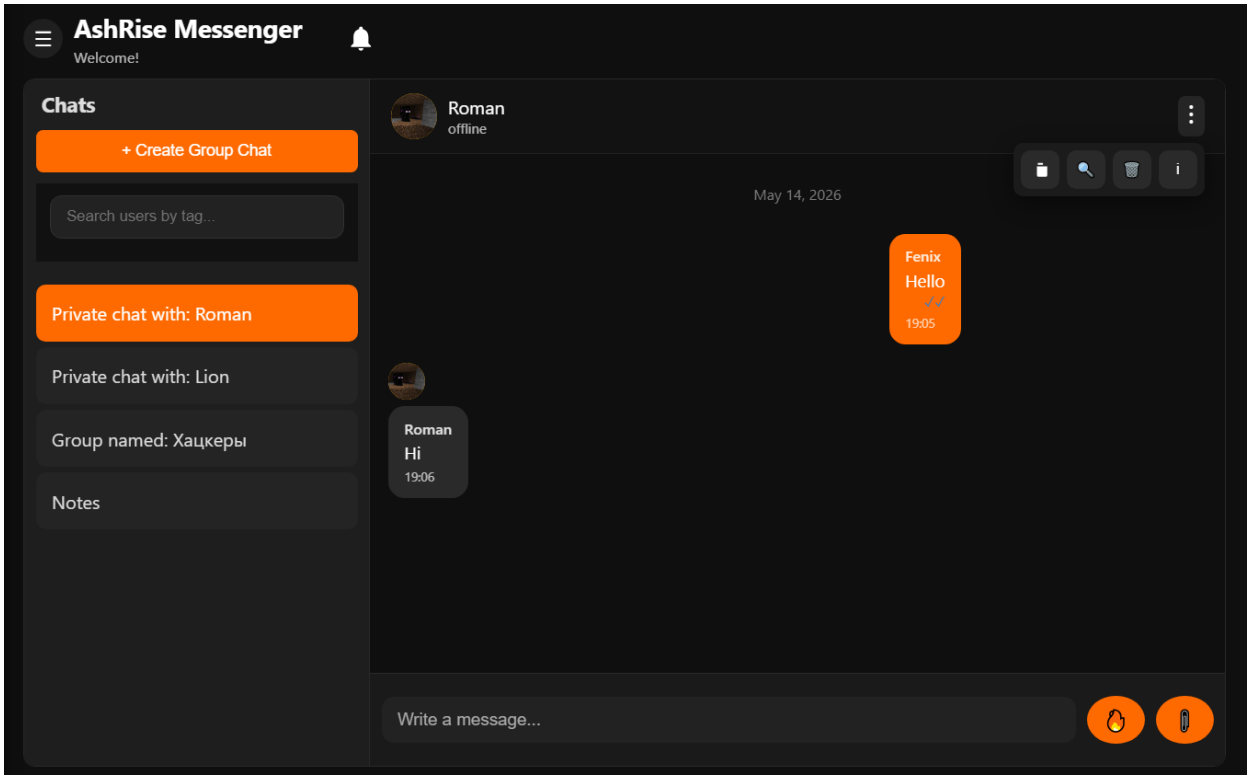


Рисунок 3.21 – Меню дій з приватним чатом

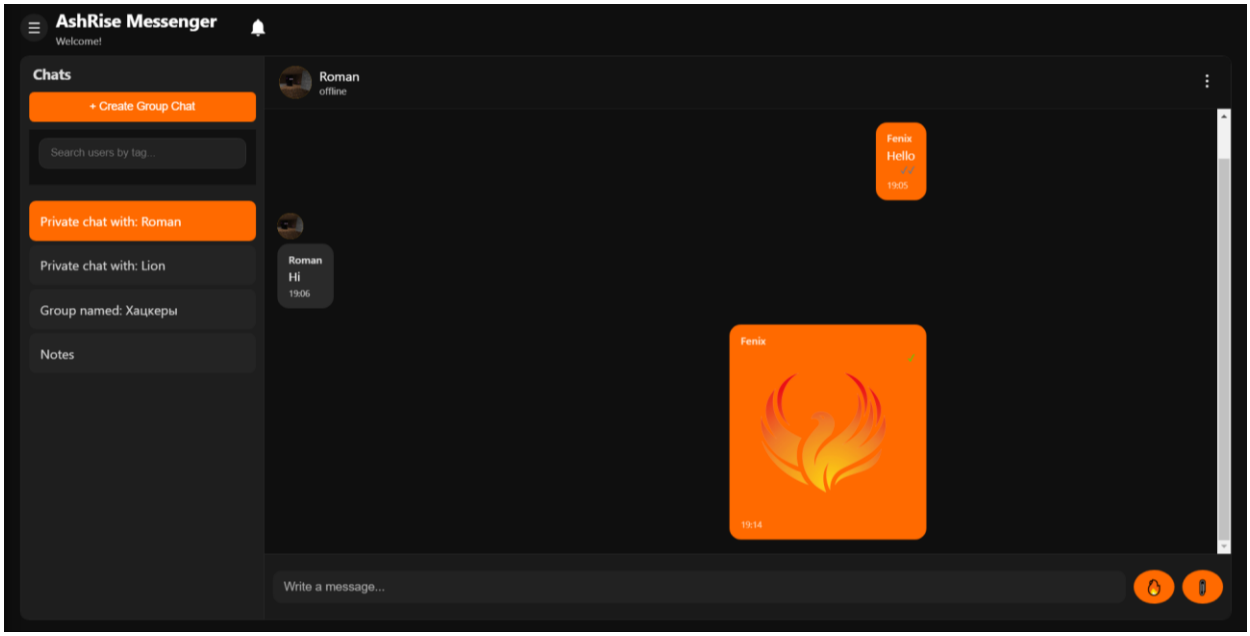


Рисунок 3.22 – Відправка вибраного фото

3.2.4 Створення та керування групами

Для колективного спілкування в застосунку реалізовано функціонал групових чатів. Процес створення спільноти, інструменти модерації та права учасників розглядаються нижче.

Створення групового чату. Ініціація процесу відбувається за допомогою кнопки «+ Create Group Chat».

У модальному вікні, що з’являється (див. рис. 3.23), користувач вводить назву групи, за бажанням завантажує аватар та обирає учасників зі свого списку друзів.

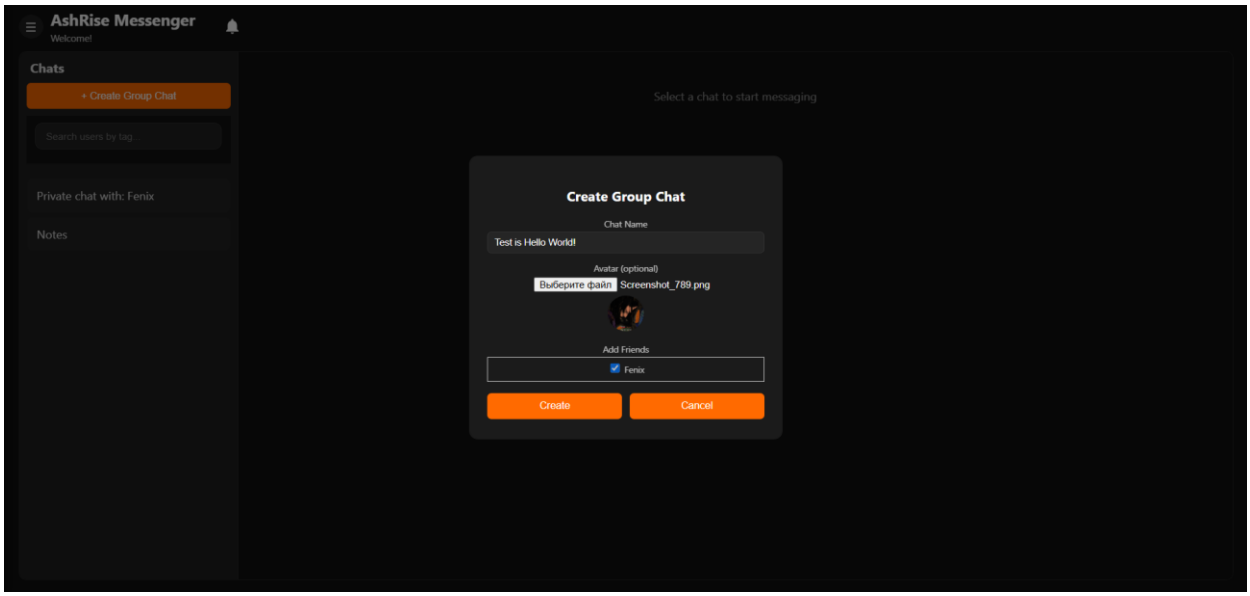


Рисунок 3.23 – Робота кнопки створення групи

Можливості адміністратора. Творець групи автоматично отримує розширені права модератора. Через картку інформації про групу (див. рис. 3.24) адмін може бачити загальну кількість учасників, їхні статуси та примусово виключати користувачів із чату за допомогою кнопки «Kick».

Можливості учасника. Для користувачів із базовою роллю «Member» функції видалення інших людей заблоковані (див. рис. 3.25). Звичайні учасники мають змогу лише переглядати список присутніх у групі, відстежувати їхню активність та самостійно залишати груповий чат.

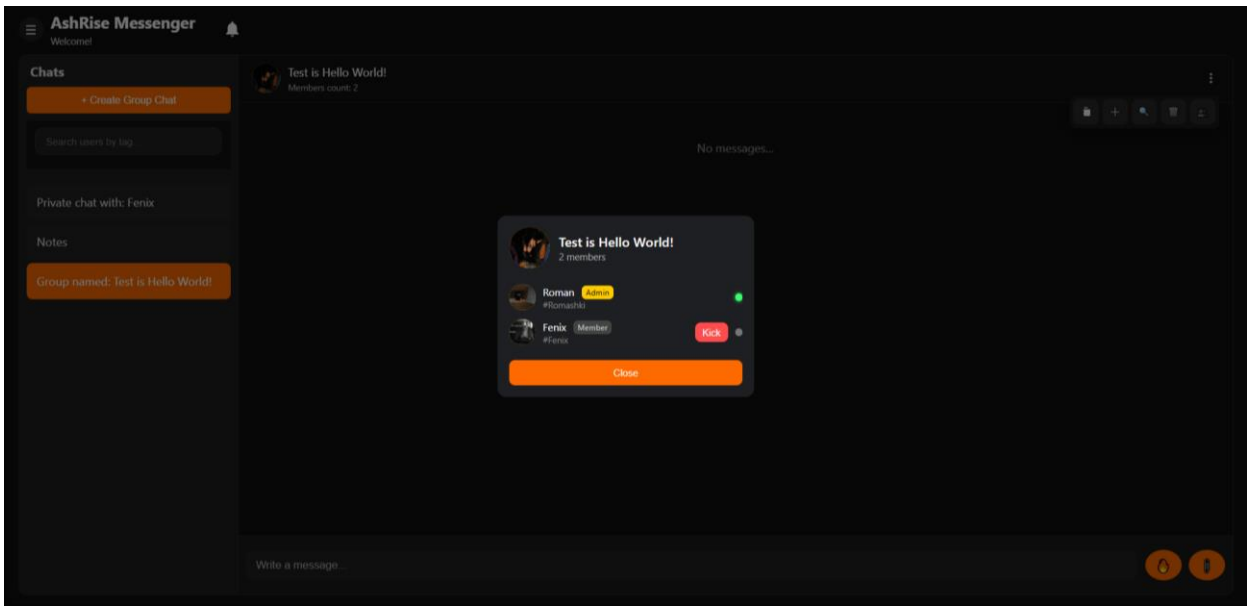


Рисунок 3.24 – Можливості адміна

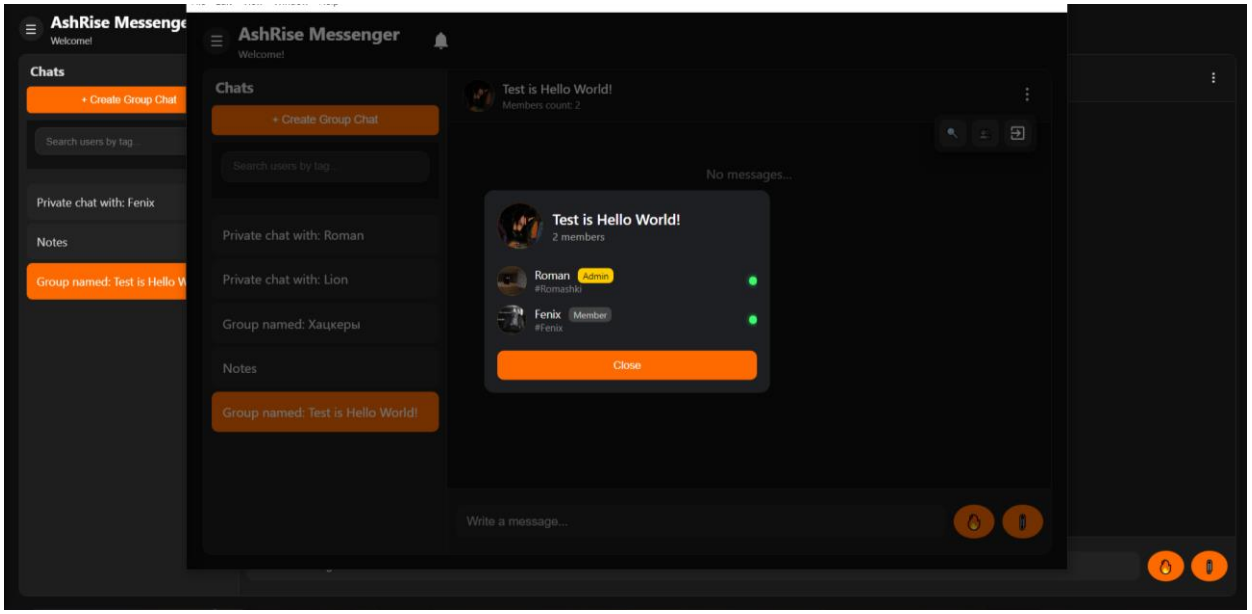


Рисунок 3.25 – Можливості учасника

Таким чином, успішна програмна реалізація всіх модулів месенджера підтвердили повну відповідність розробленого продукту висунутим технічним вимогам, забезпечивши його стабільну та надійну роботу.

ВИСНОВКИ

У кваліфікаційній роботі успішно розв’язано актуальне науково-практичне завдання, пов’язане з проєктуванням та програмною реалізацією кросплатформеної інформаційної системи «AshRise Messenger». У процесі дослідження було проаналізовано сучасні підходи до побудови систем миттєвого обміну повідомленнями.

На основі аналізу сучасних підходів та аналогів у сфері систем обміну миттєвими повідомленнями виявлено брак гнучких рішень із локальним контролем даних. Обґрунтовано вибір гібридного технологічного стеку, що поєднує фреймворк Electron, мову JavaScript (Node.js) та реляційну СУБД MySQL. Даний підхід дозволив об’єднати швидкість побудови інтерфейсу за допомогою вебтехнологій із можливостями нативного десктопного застосунку та прямим доступом до ресурсів операційної системи.

Під час проєктування архітектури месенджера розроблено стійку клієнт-серверну модель взаємодії компонентів. Спроєктовано та реалізовано централізовану схему бази даних MySQL для збереження облікових записів, статусів системи дружби та конфігурацій групових чатів.

У ході програмної реалізації десктопного клієнта в середовищі Electron розроблено модульну структуру застосунку із чітким розділенням основного процесу й процесів рендерингу. Використовуючи безпечний міст contextBridge та технологію міжпроцесної взаємодії (IPC), реалізовано унікальну підсистему кастомних сповіщень через ізольоване вікно notification.html. Це забезпечило можливість миттєвого читання, обробки та переходу до чатів безпосередньо поверх нативного інтерфейсу Windows без додаткового навантаження на головний потік програми.

ПЕРЕЛІК ПОСИЛАНЬ

1. Instant messaging. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/Миттєвий_обмін_повідомленнями (дата звернення: 15.01.2026).

2. What is instant messaging? *IBM*. URL: https://www.ibm.com/topics/instant-messaging (дата звернення: 15.01.2026).

3. Telegram Official Website. URL: https://telegram.org/ (дата звернення: 15.01.2026).

4. WhatsApp Official Website. URL: https://www.whatsapp.com/ (дата звернення: 15.01.2026).

5. Signal Official Website. URL: https://signal.org/ (дата звернення: 15.01.2026).

6. Viber Official Website. URL: https://www.viber.com/ (дата звернення: 15.01.2026).

7. The problem with modern messaging apps. *Wired*. URL: https://www.wired.com/story/how-messaging-apps-took-over-the-world/ (дата звернення: 17.01.2026).

8. Spam and phishing in messaging apps. *Kaspersky*. URL: https://www.kaspersky.com/resource-center/threats/spam-and-phishing (дата звернення: 17.01.2026).

9. Monolithic vs. microservices architecture. *Microsoft Azure*. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices (дата звернення: 17.01.2026).

10. Electron Documentation. *Electron*. URL: https://www.electronjs.org/docs/latest (дата звернення: 19.01.2026).

11. Express – Node.js web application framework. URL: https://expressjs.com/ (дата звернення: 19.01.2026).