НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри комп'ютерних наук**
_____ **Ігор БОЛБОТ**	_____ **Белла ГОЛУБ**
(підпис)	(підпис)
«___»_____ 2026 р.	«___»_____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Програмне забезпечення системи спілкування за архітектурою клієнт-сервер»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи _____ **Максим НЕДЬОШЕВ**
(підпис)

Консультант бакалаврської кваліфікаційної роботи _____ **Роман РУДЕНСЬКИЙ**
д.е.н., професор (підпис)

Виконав _____ **Юрій ПОПАТЕНКО**
(підпис)

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Попатенку Юрію Миколайовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи

«Програмне забезпечення системи спілкування за архітектурою клієнт-сервер»

затверджена наказом від «19» грудня 2025 р. № 3196 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Дослідження порівняння алгоритмів шифрування
Перелік питань, що підлягають дослідженню:
Аналіз предметної області, проектування і розробка програмного забезпечення.
Перелік графічних документів (за необхідності).
_____
Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Максим НЕДЬОШЕВ**

Консультант бакалаврської кваліфікаційної роботи _____ (підпис) **Роман РУДЕНСЬКИЙ**
д.е.н., професор

Завдання взяв до виконання _____ (підпис) **Юрій ПОПАТЕНКО**

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СУБД - система управління базами даних

API - Application Programming Interface — інтерфейс прикладного програмування

CASE - Computer-Aided Software Engineering — автоматизоване проектування програмного забезпечення

ER - Entity-Relationship — сутність-зв'язок

GCC - GNU Compiler Collection — колекція компіляторів GNU

GNU - GNU's Not Unix — рекурсивна абревіатура; основа відкритого програмного середовища

HTTP - HyperText Transfer Protocol — протокол передавання гіпертексту

IP - Internet Protocol — інтернет-протокол

JSON - JavaScript Object Notation — текстовий формат обміну даними

JWT - JSON Web Token — токен авторизації на основі JSON

MIME - Multipurpose Internet Mail Extensions — багатоцільові розширення пошти Інтернет

RFC - Request for Comments — офіційний технічний документ стандартів Інтернет

SOA - Service-Oriented Architecture — сервісно-орієнтована архітектура

SQL - Structured Query Language — мова структурованих запитів

SSL - Secure Sockets Layer — рівень захищених сокетів

SMTP - Simple Mail Transfer Protocol — протокол простої передачі пошти

SMTPS - Simple Mail Transfer Protocol Secure — захищений протокол передачі пошти

SSD - Solid State Drive — твердотільний накопичувач

TCP - Transmission Control Protocol — протокол керування передаванням

TLS - Transport Layer Security — безпека транспортного рівня

UML - Unified Modeling Language — уніфікована мова моделювання

URL - Uniform Resource Locator — уніфікований локатор ресурсу

ВСТУП

Сучасний розвиток інформаційних технологій та цифрових комунікацій суттєво змінив способи взаємодії користувачів у мережі. Одним із найпоширеніших засобів комунікації стали системи миттєвого обміну повідомленнями, які забезпечують швидку передачу текстових повідомлень, файлів, мультимедійного контенту та підтримують взаємодію між користувачами у режимі реального часу. Месенджери активно використовуються як у повсякденному спілкуванні, так і у професійній діяльності, навчанні та командній роботі. У зв’язку з цим зростає потреба у створенні сучасних, безпечних та функціональних програмних систем для організації онлайн-комунікації.

Однією з важливих вимог до сучасних систем обміну повідомленнями є забезпечення стабільної та ефективної роботи серверної частини при великій кількості одночасних користувачів. Сервер повинен коректно обробляти мережеві запити, підтримувати постійне з’єднання між клієнтом і сервером та забезпечувати надійну доставку повідомлень. Для цього необхідно використовувати ефективні механізми обробки мережевих подій, оптимізовану архітектуру клієнт-серверної взаємодії та сучасні підходи до керування ресурсами системи [1].

Ще однією важливою задачею є забезпечення захисту персональних даних користувачів та конфіденційності листування. Передача повідомлень через мережу Інтернет супроводжується ризиками перехоплення даних або несанкціонованого доступу до інформації. Тому сучасні програмні системи обміну повідомленнями повинні використовувати захищені мережеві протоколи та механізми шифрування з’єднання. Використання TLS-з’єднання дозволяє забезпечити безпечну передачу повідомлень між клієнтом та сервером і мінімізувати ризики витоку інформації.

Важливою складовою сучасних месенджерів є підтримка розширених функціональних можливостей. Користувачі очікують не лише текстового спілкування, а й можливості надсилання файлів, створення групових чатів, редагування та видалення повідомлень, перегляду статусу користувачів, налаштування профілю та використання зручного графічного інтерфейсу. Реалізація таких функцій потребує використання сучасних підходів до проєктування клієнт-серверних систем та ефективної взаємодії між різними компонентами програмного забезпечення.

Окремою задачею є підтримка масштабованості та стабільності роботи сервера. У багатокористувацьких системах сервер повинен одночасно обслуговувати велику кількість клієнтів, забезпечувати обробку мережевих подій та зберігання інформації у базі даних. Для цього використовуються сучасні технології асинхронної обробки мережевих з’єднань, механізми connection pool для роботи з базами даних та системи ефективного керування ресурсами сервера.

Сучасні технології дозволяють вирішувати зазначені задачі шляхом створення багатофункціональних клієнт-серверних систем обміну повідомленнями. Використання мови програмування C++, бібліотеки Qt для створення графічного інтерфейсу, PostgreSQL для зберігання даних та OpenSSL для захисту мережевого з’єднання дозволяє реалізувати продуктивний та безпечний програмний застосунок для онлайн-комунікації користувачів.

Незважаючи на велику кількість існуючих месенджерів, багато з них мають обмеження, пов’язані із закритістю програмної архітектури, високими вимогами до системних ресурсів або складністю адаптації під конкретні потреби користувачів. Тому розробка власного програмного забезпечення для організації захищеного обміну повідомленнями є актуальним та важливим завданням у сфері програмної інженерії.

Аналіз предметної області показує, що створення сучасного клієнт-серверного месенджера дозволяє забезпечити ефективну взаємодію

користувачів, підвищити рівень безпеки передачі даних, реалізувати підтримку мультимедійних повідомлень та оптимізувати процес онлайн-комунікації. Це підтверджує актуальність теми дослідження та обґрунтовує необхідність розробки програмного забезпечення для системи обміну повідомленнями [2].

Предметом цієї бакалаврської роботи є процес організації захищеного обміну повідомленнями між користувачами у клієнт-серверній інформаційній системі. Система обміну повідомленнями включає програмні засоби для реєстрації та авторизації користувачів, передачі повідомлень у режимі реального часу, створення групових чатів, надсилання файлів та керування профілем користувача.

Підтримка комунікації між користувачами передбачає взаємодію між клієнтською та серверною частинами системи. Клієнтський застосунок забезпечує графічний інтерфейс користувача, обробку введених даних та відображення повідомлень. Серверна частина відповідає за обробку мережевих з’єднань, маршрутизацію повідомлень між користувачами, роботу з базою даних та забезпечення безпеки передачі інформації.

Предметною областю також є використання сучасних технологій для створення програмного забезпечення системи обміну повідомленнями. Для реалізації клієнтської частини використовується бібліотека Qt, яка дозволяє створювати сучасний графічний інтерфейс користувача та підтримувати TLS-з’єднання через QSslSocket. Серверна частина реалізується мовою програмування C++ із використанням бібліотек OpenSSL та PostgreSQL для організації безпечного зберігання та передачі даних.

Важливою частиною предметної області є робота з базою даних. Система повинна забезпечувати зберігання інформації про користувачів, чати, повідомлення, вкладення, групові бесіди та токени авторизації. Використання PostgreSQL дозволяє реалізувати надійне та масштабоване зберігання даних, а також підтримувати ефективну роботу серверної частини застосунку.

Предметна область також охоплює реалізацію сучасного користувацького інтерфейсу месенджера. Інтерфейс повинен забезпечувати зручну навігацію між чатами, підтримку пошуку контактів, відображення статусу користувачів, роботу з повідомленнями та файлами, а також підтримку редагування та видалення повідомлень у реальному часі.

1 ОБМІН ПОВІДОМЛЕННЯМИ ЯК ОБ’ЄКТ ДОСЛІДЖЕННЯ

1.1 Аналіз системи обміну повідомленнями як предметної області

Система обміну повідомленнями в реальному часі (месенджер) — це програмний комплекс, що забезпечує миттєву комунікацію між користувачами через мережу Інтернет або локальну мережу. Основними функціями такої системи є реєстрація та автентифікація користувачів, організація приватних і групових розмов, обмін текстовими повідомленнями та файлами, а також відстеження статусу присутності учасників у режимі реального часу. Подібні системи широко застосовуються як у побутовому спілкуванні, так і в корпоративному середовищі для координації роботи команд та оперативного вирішення питань [3].

Одним із ключових етапів взаємодії користувача із системою є реєстрація та автентифікація. Під час першого звернення до застосунку користувач проходить процедуру реєстрації, в ході якої створюється обліковий запис. До нього вносяться основні дані: електронна адреса, нікнейм та пароль. Пароль не зберігається у відкритому вигляді — він хешується з використанням алгоритму Argon2id, що унеможливлює його відновлення навіть у разі компрометації бази даних [4]. Надалі при кожному вході до системи користувач проходить автентифікацію, а у разі втрати пароля передбачено механізм його відновлення через підтвердження шестизначним кодом, надісланим на електронну пошту.

Обслуговування користувача передбачає певний сценарій взаємодії із системою: від авторизації — до встановлення з'єднання, завантаження списку чатів та обміну повідомленнями. Після успішної автентифікації клієнтський застосунок отримує актуальний список контактів і чатів, а сервер фіксує

користувача як онлайн і сповіщає про це його співрозмовників. Усі дані передаються виключно через захищений канал із використанням протоколу TLS, що забезпечує конфіденційність переданої інформації та захист від перехоплення [5].

Організація чатів є одним із найважливіших процесів у системі. Користувач може ініціювати приватну розмову з іншим зареєстрованим користувачем, знаходячи його за електронною адресою. Крім того, підтримується створення групових чатів, до яких можна залучити кількох учасників одночасно, зазначивши назву групи та список членів. На рівні сервера система перевіряє наявність відповідного запису в базі даних та автоматично повідомляє усіх онлайн-учасників про створення нового чату [6].

Обмін повідомленнями та файлами є центральною функцією системи. Текстові повідомлення передаються у форматі JSON через постійне TLS-з'єднання та зберігаються у базі даних із фіксацією часу відправлення. Система підтримує редагування та видалення надісланих повідомлень, причому усі зміни негайно відображаються у всіх учасників чату завдяки механізму широкомовного розсилання повідомлень. Передача файлів реалізована через кодування вмісту у форматі Base64 та збереження файлів на сервері, звідки будь-який учасник чату може їх завантажити у зручний для нього момент [7].

Інформація про користувачів та їхню активність має бути структурованою та актуальною. Зокрема, важливо мати змогу в реальному часі відображати статус присутності співрозмовника — «в мережі» або «не в мережі», бачити повну історію переписки при відкритті чату, а також переглядати профіль іншого учасника, включно з його аватаром та нікнеймом. Це дозволяє користувачам приймати зважені рішення щодо комунікації та покращує загальний досвід взаємодії із застосунком. Для власного профілю передбачена можливість редагування особистих даних та завантаження фотографії аватара.

Окрім безпосереднього обміну повідомленнями, у системі здійснюється детальний облік технічних даних. На рівні сервера ведеться управління пулом з'єднань із базою даних, відстеження активних клієнтських сесій та асинхронна обробка мережевих подій. Це дозволяє контролювати навантаження на серверну інфраструктуру та забезпечувати стабільну роботу застосунку при одночасній роботі багатьох користувачів. Збереження всіх повідомлень, файлів та метаданих у реляційній базі даних дозволяє формувати повну картину взаємодій та забезпечувати цілісність інформації навіть при раптовому відключенні клієнта від мережі [8].

Таким чином, система обміну повідомленнями в реальному часі — це складний багатокомпонентний програмний комплекс, у якому взаємодіють користувачі, клієнтський застосунок, серверна частина, база даних та зовнішні сервіси електронної пошти. Автоматизація процесів автентифікації, маршрутизації повідомлень, управління чатами та захисту даних є необхідною умовою для створення зручного, надійного та безпечного інструменту комунікації.

1.2 Моделювання предметної області

Unified Modeling Language (UML) є універсальною мовою моделювання програмних систем, яка використовується для опису структури, поведінки та архітектури програмного забезпечення. UML надає стандартизований підхід до візуалізації програмних рішень та дозволяє формалізувати процес проєктування системи. Використання UML значно спрощує взаємодію між розробниками, аналітиками та іншими учасниками процесу створення програмного забезпечення, оскільки забезпечує єдине представлення компонентів системи та логіки їх взаємодії.

Основною перевагою UML є можливість одночасного моделювання як структурної, так і поведінкової частини програмного продукту. За допомогою UML можна описати внутрішню будову системи, зв’язки між її

компонентами, логіку виконання процесів та сценарії взаємодії користувачів із програмним застосунком. Це дозволяє ще на етапі проєктування виявити потенційні проблеми архітектури, оптимізувати структуру системи та підвищити якість майбутнього програмного забезпечення.

UML містить набір різних типів діаграм, які поділяються на структурні та поведінкові. До структурних діаграм належать діаграми класів, компонентів, об’єктів, пакетів та розгортання. Вони використовуються для опису архітектури системи, структури даних та взаємозв’язків між окремими елементами. Поведінкові діаграми, такі як діаграми прецедентів, послідовностей, діяльності та станів, дозволяють моделювати логіку роботи системи, порядок виконання операцій та взаємодію між користувачами і програмними компонентами.

У межах розробки програмної системи обміну повідомленнями UML використовується для моделювання клієнт-серверної архітектури, механізмів автентифікації користувачів, обробки повідомлень та взаємодії між окремими модулями системи. Діаграми прецедентів дозволяють визначити основні сценарії роботи користувача із системою, зокрема реєстрацію, авторизацію, створення чатів, надсилання повідомлень та перегляд історії листування. Це дає можливість формалізувати функціональні вимоги системи та визначити основні ролі користувачів.

Діаграми класів використовуються для опису структури бази даних та взаємозв’язків між основними сутностями системи, такими як користувачі, повідомлення, чати, токени авторизації та серверні сесії. Таке моделювання дозволяє забезпечити цілісність даних, уникнути дублювання інформації та оптимізувати процес збереження повідомлень у базі даних PostgreSQL.

Діаграми послідовностей демонструють процес взаємодії між клієнтською та серверною частинами застосунку. За допомогою цих діаграм можна відобразити послідовність передачі повідомлень, встановлення TLS-з’єднання, процес автентифікації користувача та механізм оновлення refresh-

токенів. Це дозволяє детально проаналізувати роботу системи та визначити критичні точки взаємодії між її компонентами.

Крім того, UML дозволяє моделювати виняткові ситуації та сценарії обробки помилок, що є важливим для забезпечення стабільної роботи системи обміну повідомленнями. Діаграми діяльності використовуються для аналізу логіки обробки повідомлень та маршрутизації запитів між сервером і клієнтами. Діаграми станів дозволяють відстежувати зміни станів користувача, наприклад авторизований, неавторизований або відключений від сервера.

Використання UML у процесі розробки системи обміну повідомленнями забезпечує низку переваг, серед яких покращення документування програмного забезпечення, спрощення командної роботи, підвищення зрозумілості архітектури системи та полегшення подальшої підтримки й масштабування застосунку. Завдяки UML стає можливим створення надійної, безпечної та масштабованої клієнт-серверної системи обміну повідомленнями, здатної забезпечити стабільну взаємодію користувачів у режимі реального часу.

1.2.1 Діаграма прецедентів. Діаграма прецедентів (Use-Case Diagram) є одним із основних інструментів UML, який використовується для моделювання функціональних можливостей програмної системи та взаємодії користувачів із нею. Основною метою діаграми прецедентів є наочне представлення функцій системи, а також визначення ролей користувачів і сценаріїв їхньої взаємодії із програмним забезпеченням. Такий підхід дозволяє ще на етапі проєктування формалізувати вимоги до системи, визначити межі функціоналу та забезпечити зрозумілу структуру взаємодії між клієнтом і серверною частиною застосунку.

У контексті розробки клієнт-серверного чат-застосунку діаграма прецедентів дозволяє описати основні сценарії роботи користувача із системою обміну повідомленнями. Система забезпечує реєстрацію та авторизацію користувачів, створення чатів, обмін повідомленнями у режимі

реального часу, керування профілем користувача та підтримку безпечної автентифікації за допомогою JWT і refresh-токенів. Використання UML у даному випадку дозволяє формалізувати всі основні процеси взаємодії клієнта із сервером та відобразити логіку роботи програмного забезпечення.

На діаграмі прецедентів основним актором виступає користувач системи — «Клієнт». Саме він взаємодіє із програмою через графічний інтерфейс Qt-застосунку та виконує всі основні дії, пов’язані із використанням чат-системи. Сервер у даній архітектурі виконує роль компонента обробки запитів, автентифікації користувачів, управління сесіями та маршрутизації повідомлень між клієнтами.

Розроблена діаграма прецедентів представлена на рис. 1.1

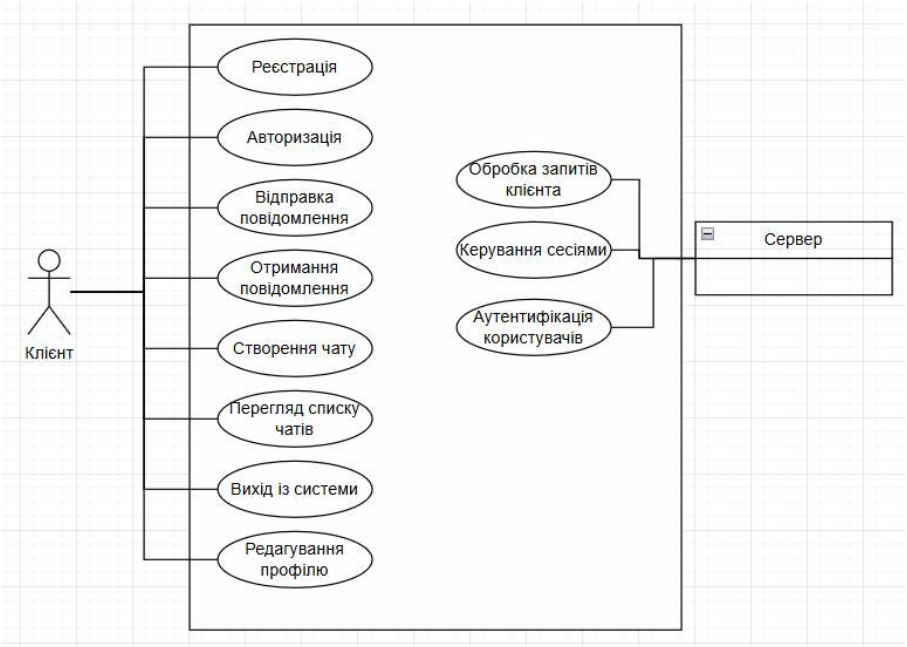

Рис. 1.1 – Діаграма прецедентів системи обміну повідомленнями

Створена діаграма містить таких учасників:

- «Клієнт»;
- «Сервер».

Актор «Клієнт» включає такі прецеденти:

- реєстрація;
- авторизація;
- відправка повідомлення;
- отримання повідомлення;
- створення чату;
- перегляд списку чатів;
- вихід із системи;
- редагування профілю.

Серверна частина системи включає такі функціональні прецеденти:

- обробка запитів клієнта;
- керування сесіями;
- автентифікація користувачів.

Прецеденти взаємопов’язані між собою та утворюють єдиний процес функціонування клієнт-серверного чат-застосунку. Наприклад, виконання прецеденту «Відправка повідомлення» можливе лише після успішної авторизації користувача, а прецедент «Керування сесіями» забезпечує підтримку активних JWT-сесій та механізм оновлення access-токенів через refresh-токени.

Розглянемо детальніше основні сценарії використання системи.

Варіант використання: Авторизація

Актор: Клієнт

Опис:

Варіант використання «Авторизація» описує процес входу користувача до системи обміну повідомленнями за допомогою логіна та пароля. Після успішної перевірки введених даних сервер генерує JWT

access-токен та refresh-токен, які використовуються для підтримки безпечної сесії користувача. Це дозволяє забезпечити захищений доступ до функціоналу системи та уникнути повторного введення облікових даних під час роботи із застосунком.

Основний потік:

1. користувач відкриває форму авторизації;
2. система відображає поля для введення логіна та пароля;
3. користувач вводить облікові дані;
4. клієнтський застосунок надсилає запит на сервер;
5. сервер перевіряє правильність введених даних у базі PostgreSQL;
6. у випадку успішної перевірки сервер генерує JWT access-токен та refresh-токен;
7. токени передаються клієнтському застосунку;
8. користувач отримує доступ до основного функціоналу системи.

Альтернативні потоки:

1. якщо логін або пароль введено неправильно, система відображає повідомлення про помилку авторизації;
2. якщо сервер недоступний, клієнт отримує повідомлення про неможливість встановлення з’єднання;
3. якщо refresh-токен втратив актуальність, система автоматично перенаправляє користувача на повторну авторизацію;
4. користувач може скасувати процес авторизації та повернутися до стартового вікна.

Використання JWT-автентифікації дозволяє забезпечити високий рівень безпеки системи, мінімізувати ризик несанкціонованого доступу та підвищити стабільність роботи клієнт-серверного застосунку.

Варіант використання: Відправка повідомлення

Актор: Клієнт

Опис:

Варіант використання «Відправка повідомлення» описує процес обміну текстовими повідомленнями між користувачами системи. Після створення або відкриття чату користувач може вводити текст повідомлення та надсилати його іншому учаснику. Сервер обробляє повідомлення, зберігає його у базі даних та передає отримувачу у режимі реального часу.

Основний потік:

1. користувач відкриває список чатів;
2. система відображає доступні діалоги;
3. користувач обирає необхідний чат;
4. система відкриває вікно переписки;
5. користувач вводить текст повідомлення;
6. повідомлення надсилається на сервер;
7. сервер перевіряє дійсність сесії користувача;
8. сервер зберігає повідомлення у базі даних;
9. повідомлення передається іншому користувачу;
10. клієнт отримує підтвердження успішної доставки повідомлення.

Альтернативні потоки:

1. якщо відсутнє з’єднання із сервером, система повідомляє про помилку надсилання;
2. якщо access-токен прострочений, виконується автоматичне оновлення через refresh-токен;
3. якщо чат не існує, сервер повертає повідомлення про помилку;
4. користувач може скасувати введення повідомлення до моменту надсилання.

Реалізація цього сценарію забезпечує швидкий та безпечний обмін повідомленнями між користувачами, а також підтримує синхронізацію чатів у режимі реального часу.

Варіант використання: Створення чату

Актор: Клієнт

Опис:

Варіант використання «Створення чату» описує процес створення нового діалогу між користувачами системи. Користувач може обрати співрозмовника зі списку доступних користувачів та ініціювати нову переписку.

Основний потік:

1. користувач відкриває меню створення чату;
2. система відображає список користувачів;
3. користувач обирає співрозмовника;
4. клієнтський застосунок надсилає запит на сервер;
5. сервер створює новий чат у базі даних;
6. система відкриває створений діалог;
7. користувач може розпочати обмін повідомленнями.

Альтернативні потоки:

1. якщо чат із даним користувачем уже існує, система відкриває наявний діалог;
2. якщо сервер недоступний, створення чату переривається;
3. якщо користувач не авторизований, система перенаправляє його на форму входу.

Використання сценарію «Створення чату» забезпечує гнучкість взаємодії між користувачами та дозволяє швидко організовувати персональні діалоги всередині системи.

1.2.2 Діаграма послідовності. Діаграма послідовності (Sequence Diagram) є одним із основних інструментів UML, який використовується для моделювання взаємодії між компонентами системи у часовому

порядку. Вона дозволяє відобразити процес обміну повідомленнями між користувачами, клієнтськими застосунками, сервером та іншими компонентами системи під час виконання певного сценарію. Основною перевагою діаграми послідовності є можливість наочного представлення логіки роботи програмного забезпечення, а також демонстрація послідовності обробки запитів і відповідей між елементами системи [9].

У контексті розробки клієнт-серверного чат-застосунку діаграма послідовності дозволяє деталізувати процеси авторизації користувачів, передачі повідомлень, взаємодії з сервером та базою даних, а також механізм доставки повідомлень іншим користувачам системи. Вона допомагає чітко зрозуміти структуру взаємодії між клієнтською частиною, серверною логікою та системою зберігання даних.

На діаграмі послідовності зазвичай виділяють такі основні компоненти:

- актор — користувач системи, який ініціює взаємодію;
- клієнтський застосунок — графічний інтерфейс, через який користувач працює із системою;
- сервер — компонент, що відповідає за обробку запитів, автентифікацію та маршрутизацію повідомлень;
- база даних — система зберігання інформації про користувачів, повідомлення та сесії;
- повідомлення — запити та відповіді, які передаються між компонентами;
- лінії життя — вертикальні лінії, що показують час існування об’єкта під час виконання сценарію.

Використання діаграми послідовності дозволяє проаналізувати порядок взаємодії компонентів системи, оцінити ефективність обробки запитів та виявити можливі проблеми у логіці функціонування клієнт-серверного застосунку. Такі діаграми також значно спрощують процес документування програмного забезпечення та полегшують подальшу реалізацію і тестування системи.

Діаграма послідовності, зображена на рис. 1.2, включає такі об’єкти:

- «Користувач»;
- «Клієнтський додаток»;
- «Сервер»;
- «База даних»;
- «Інший користувач».

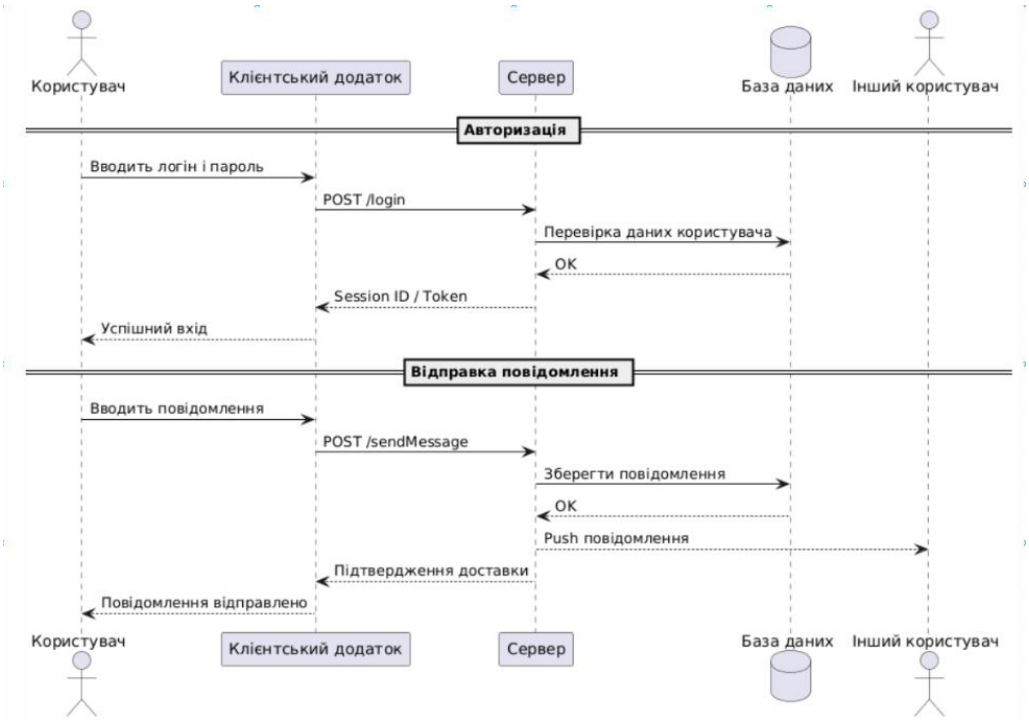

Рис. 1.2 – Діаграма послідовності

Представлена діаграма послідовності демонструє два основні сценарії роботи системи: авторизацію користувача та процес відправки повідомлення між користувачами чат-застосунку.

Перший сценарій описує процес авторизації користувача у системі. Спочатку користувач вводить логін і пароль у клієнтському додатку. Після цього клієнтський застосунок формує HTTP-запит типу POST /login та надсилає його на сервер. Сервер, у свою чергу, виконує перевірку введених даних через базу даних користувачів. Якщо облікові дані правильні, база даних повертає підтвердження успішної перевірки, після чого сервер генерує JWT-токен і передає його клієнтському застосунку. Після

отримання токена користувач успішно авторизується та отримує доступ до функціоналу системи.

Даний сценарій демонструє використання механізму автентифікації користувачів, який забезпечує безпечний доступ до чат-системи. Крім того, у системі реалізована підтримка refresh-токенів, що дозволяє автоматично оновлювати access-токени без повторного введення логіна та пароля користувачем.

Другий сценарій описує процес надсилання повідомлення між користувачами системи. Користувач вводить текст повідомлення у клієнтському застосунку, після чого застосунок надсилає POST-запит /sendMessage на сервер. Сервер приймає повідомлення та виконує його збереження у базі даних. Після успішного запису база даних повертає підтвердження серверу, а сервер здійснює push-передачу повідомлення іншому користувачу системи.

Після успішної доставки сервер надсилає клієнтському застосунку підтвердження доставки повідомлення, а користувач отримує інформацію про успішне відправлення повідомлення. Таким чином забезпечується взаємодія між користувачами у режимі реального часу.

Представлена діаграма послідовності демонструє логіку роботи клієнт-серверної архітектури чат-застосунку, де сервер виступає центральним елементом обробки запитів, управління сесіями та маршрутизації повідомлень. Використання такої архітектури дозволяє забезпечити безпечний обмін даними, підтримку активних сесій користувачів та стабільну роботу системи обміну повідомленнями.

1.2.3 Діаграма активності. Діаграма активності використовується для моделювання логіки роботи системи та відображення послідовності виконання дій між користувачем і серверною частиною застосунку. Вона дозволяє наочно показати порядок виконання операцій, можливі варіанти розвитку процесу та обробку помилок під час взаємодії із системою. На відміну від діаграми послідовності, яка акцентує увагу на обміні

повідомленнями між об’єктами, діаграма активності демонструє саме алгоритм роботи системи та переходи між окремими етапами процесу.

У контексті розробленого месенджера діаграма активності відображає процес надсилання повідомлення між користувачами. Вона показує повний цикл роботи системи: від моменту відкриття чату користувачем до підтвердження успішної доставки повідомлення або виникнення помилки під час обробки даних сервером. Такий підхід дозволяє краще зрозуміти логіку функціонування клієнт-серверної взаємодії у застосунку.

Діаграма діяльності починається з початкового вузла, після чого користувач відкриває чат та вводить текст повідомлення. Далі повідомлення надсилається на сервер, де відбувається його обробка. На етапі обробки сервер перевіряє коректність отриманих даних та визначає подальший сценарій виконання процесу. Якщо обробка проходить успішно, повідомлення доставляється отримувачу, після чого сервер надсилає підтвердження доставки, а у вікні чату користувача відображається статус «Надіслано». У випадку виникнення помилки система переходить до альтернативного потоку виконання та відображає повідомлення про помилку користувачу.

Діаграма також демонструє використання умовного вузла для реалізації розгалуження процесу залежно від результату обробки повідомлення сервером. Це дозволяє відобразити як основний сценарій успішної доставки повідомлення, так і альтернативний сценарій із помилкою. Завершення процесу позначається кінцевим вузлом після виконання одного з можливих потоків.

Використання діаграми активності дозволяє проаналізувати процес передачі повідомлень у системі, визначити можливі точки виникнення помилок та оцінити ефективність взаємодії між клієнтською та серверною частинами застосунку. Крім того, таке моделювання спрощує подальшу

оптимізацію логіки роботи месенджера та забезпечує більш зрозуміле представлення функціонування системи.

Розроблена діаграма активності представлена на рис. 1.3.

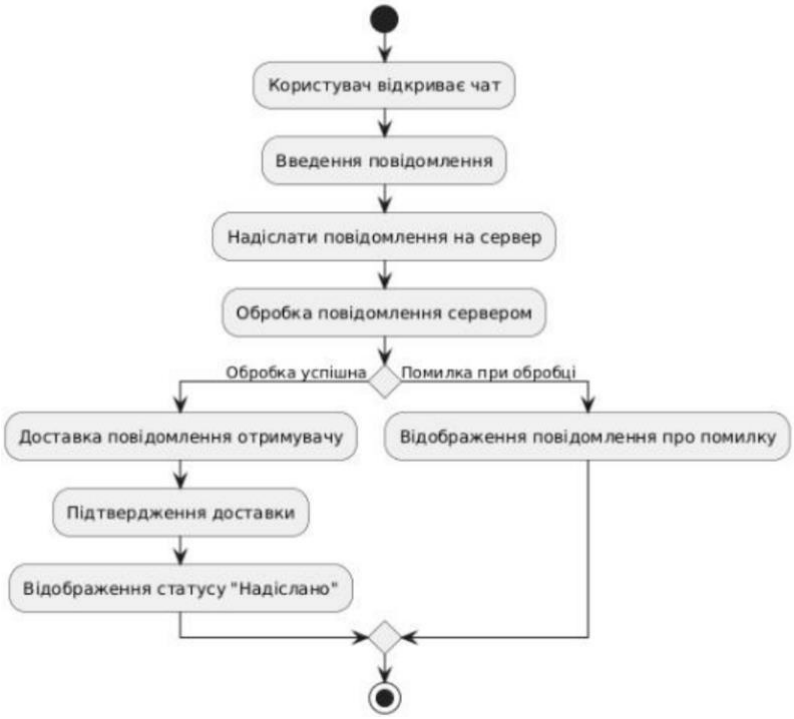

Рис. 1.3 – Діаграма активності процесу надсилання повідомлення

Дана діаграма активності демонструє логіку роботи месенджера під час надсилання повідомлення користувачем. Спочатку користувач відкриває чат і вводить текст повідомлення. Після цього повідомлення передається на сервер для подальшої обробки. Сервер перевіряє отримані дані та визначає, чи можлива успішна доставка повідомлення.

У разі успішної обробки повідомлення доставляється отримувачу, а система надсилає підтвердження доставки. Після цього у клієнтському застосунку відображається статус «Надіслано», що підтверджує успішне завершення операції. Якщо ж під час обробки виникає помилка, користувач отримує відповідне повідомлення про помилку, після чого процес завершується.

Таким чином, діаграма демонструє основний механізм обміну повідомленнями у розробленому месенджері, а також обробку можливих помилок під час взаємодії клієнта із сервером.

1.3 Огляд існуючих рішень

З розвитком цифрових комунікацій значно зросла потреба у швидкому, стабільному та захищеному обміні повідомленнями між користувачами. Сучасні месенджери використовуються не лише для особистого спілкування, але й для навчання, командної роботи, бізнес-комунікації та обміну файлами. У зв’язку з цим на ринку представлена велика кількість програмних рішень для організації онлайн-спілкування, які забезпечують передачу повідомлень у режимі реального часу, підтримують мультимедійний контент та використовують сучасні механізми захисту даних. Найбільш поширеними є десктопні месенджери, мобільні платформи для обміну повідомленнями та корпоративні системи комунікації.

Одним із найбільш популярних сучасних месенджерів є Telegram. Це багатоплатформна система миттєвого обміну повідомленнями, яка забезпечує швидку передачу текстових повідомлень, файлів, фотографій, відео та інших типів даних між користувачами. Основною перевагою Telegram є висока швидкість роботи, підтримка хмарного зберігання повідомлень та використання механізмів шифрування для захисту даних користувачів [5].

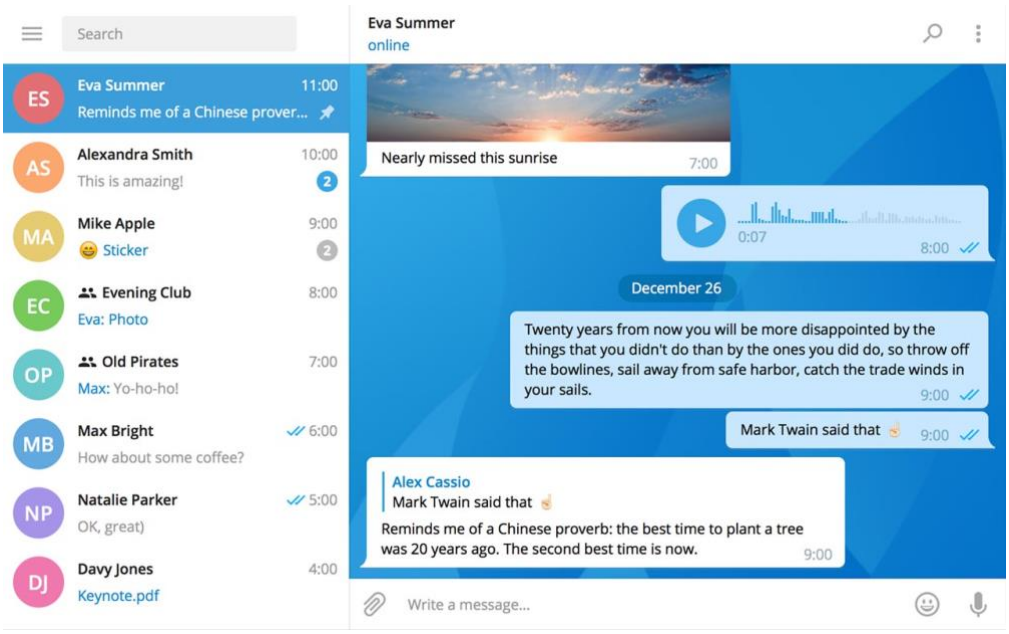

Рис. 1.4 Месенджер Telegram

Основною функцією Telegram є забезпечення швидкої та безпечної комунікації між користувачами. Платформа підтримує приватні та групові чати, канали, передачу файлів великого розміру, голосові повідомлення та відеодзвінки. Крім того, Telegram використовує власний мережевий протокол MTProto, який забезпечує високу швидкість передачі даних та захист інформації під час обміну повідомленнями.

Важливою перевагою Telegram є підтримка багатоплатформності. Користувачі можуть працювати із системою через мобільні застосунки, вебверсію або десктопний клієнт. Крім того, платформа підтримує API для розробників, що дозволяє створювати ботів, інтегрувати сторонні сервіси та реалізовувати власні рішення на основі інфраструктури Telegram.

Ще однією важливою особливістю є підтримка хмарної синхронізації повідомлень. Усі дані користувача автоматично синхронізуються між пристроями, що дозволяє отримувати доступ до історії чатів із будь-якого підключеного клієнта. Також Telegram підтримує створення групових чатів із великою кількістю учасників та передачу мультимедійного контенту без значних обмежень.

Незважаючи на значну кількість переваг, Telegram має певні обмеження. Платформа є закритою системою, а її серверна архітектура недоступна для модифікації або адаптації під індивідуальні потреби користувачів. Крім того, використання сторонньої інфраструктури не дозволяє повністю контролювати процес зберігання та передачі даних. Це створює потребу у розробці власних клієнт-серверних рішень для організації захищеного обміну повідомленнями.

Отже, Telegram є сучасним та ефективним месенджером із широким набором функціональних можливостей, який активно використовується для організації онлайн-комунікації. Проте залежність від сторонньої інфраструктури та обмежені можливості кастомізації підтверджують доцільність розробки власного програмного забезпечення для обміну повідомленнями, що є однією з основних цілей даної бакалаврської роботи.

Розглянемо інше популярне програмне рішення у сфері онлайн-комунікації.

Одним із найпоширеніших корпоративних месенджерів є Discord. Це програмна платформа для текстового, голосового та відеоспілкування, яка спочатку була орієнтована на геймерську аудиторію, але згодом стала активно використовуватись для навчання, командної роботи та організації онлайн-спільнот [10].

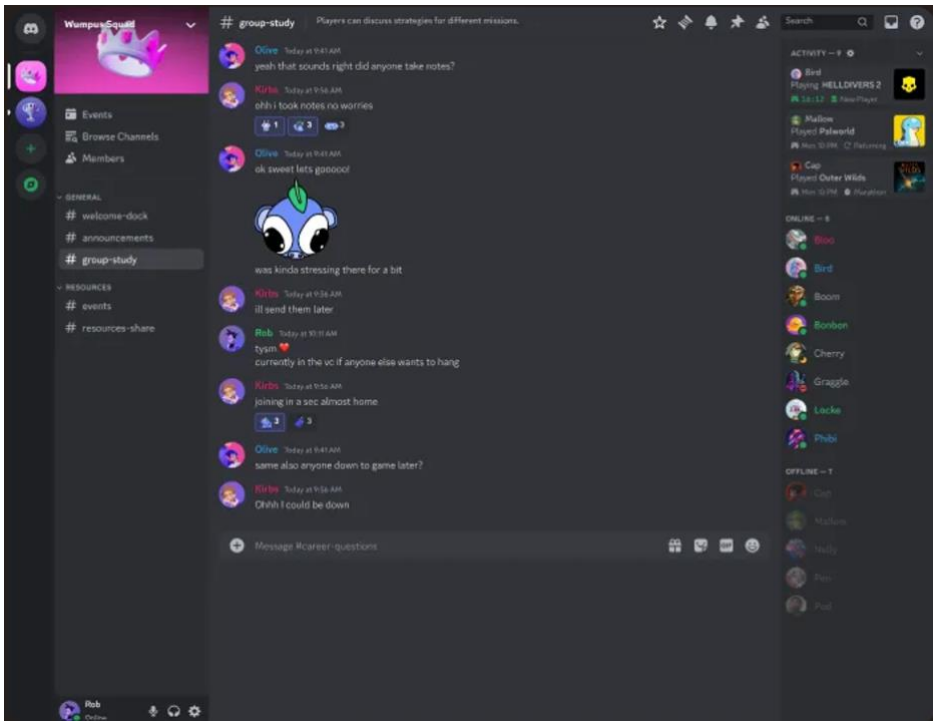

Рис. 1.5 Месенджер Discord

Основною особливістю Discord є підтримка серверної структури організації чатів. Користувачі можуть створювати власні сервери, текстові та голосові канали, керувати ролями учасників та налаштовувати права доступу. Такий підхід дозволяє організовувати ефективну комунікацію між великою кількістю користувачів у межах однієї системи.

Важливою перевагою Discord є підтримка роботи у режимі реального часу. Повідомлення передаються практично миттєво, а система забезпечує стабільне з’єднання навіть при великій кількості одночасно підключених користувачів. Платформа також підтримує надсилання файлів, використання мультимедійного контенту, інтеграцію ботів та автоматизацію окремих процесів.

Discord використовує сучасні технології клієнт-серверної взаємодії та підтримує багатоплатформність. Користувачі можуть працювати через десктопні застосунки, браузер або мобільні пристрої. Крім того, система надає API для розробників, що дозволяє створювати власних ботів та інтегрувати зовнішні сервіси.

Водночас Discord має певні недоліки. Значна кількість функцій робить систему досить складною для користувачів, які потребують лише базового обміну повідомленнями. Крім того, серверна частина повністю контролюється компанією-розробником, що обмежує можливості створення власних модифікованих рішень. Для деяких функцій також необхідне постійне підключення до Інтернету та використання значних системних ресурсів.

Таким чином, Discord є потужною платформою для організації онлайн-комунікації з підтримкою текстових, голосових та відеочатів. Незважаючи на широкі функціональні можливості, залежність від сторонньої серверної інфраструктури та надмірна складність окремих компонентів підтверджують актуальність розробки власного програмного забезпечення для системи обміну повідомленнями, яке буде більш адаптованим до конкретних вимог користувачів та забезпечуватиме повний контроль над передачею і зберіганням даних.

1.4 Постановка завдання

Організація сучасної системи миттєвого обміну повідомленнями базується на необхідності забезпечення швидкої, стабільної та захищеної комунікації між користувачами у режимі реального часу. У сучасному цифровому середовищі користувачі очікують можливості оперативного обміну текстовими повідомленнями, файлами та мультимедійним контентом із використанням зручного графічного інтерфейсу та високого рівня захисту персональних даних. Водночас розробникам програмного забезпечення необхідно забезпечити стабільність роботи системи, підтримку багатокористувацького режиму та ефективну взаємодію між клієнтською та серверною частинами застосунку.

Основним завданням системи є організація захищеного обміну повідомленнями між користувачами та забезпечення коректної роботи всіх

функціональних модулів месенджера. Система повинна підтримувати передачу повідомлень у режимі реального часу, обробку підключень користувачів, збереження історії чатів та роботу з файлами. Ключові параметри, що підлягають обробці та контролю, включають:

автентифікацію та авторизацію користувачів для забезпечення захищеного доступу до системи;

передачу текстових повідомлень між клієнтами у режимі реального часу;

- підтримку групових чатів та керування списком учасників;
- надсилання та отримання файлів різних форматів;
- збереження історії повідомлень у базі даних;
- відображення статусу користувачів у мережі;
- редагування та видалення повідомлень;
- захист мережевого з’єднання за допомогою TLS та OpenSSL [11].

Програмний застосунок підтримує централізовану базу даних PostgreSQL, яка зберігає інформацію про користувачів, чати, повідомлення, групи, файли та токени авторизації. Узгодженість і цілісність даних є критично важливими, оскільки система повинна запобігати втраті повідомлень, дублюванню даних та помилкам під час одночасної роботи великої кількості користувачів.

Процес обміну повідомленнями організований за чітко визначеним алгоритмом взаємодії між клієнтом і сервером. Після авторизації користувач встановлює захищене TLS-з’єднання із сервером. Коли користувач надсилає повідомлення, клієнтський застосунок формує JSON-запит та передає його серверу. Сервер обробляє отримані дані, виконує перевірку прав доступу, зберігає повідомлення у базі даних та перенаправляє його відповідним користувачам. Усі дії фіксуються у системі, що дозволяє підтримувати історію повідомлень та забезпечувати коректну синхронізацію чатів між клієнтами.

Інтегрований механізм комунікації дозволяє користувачам обмінюватися повідомленнями та файлами безпосередньо через програмний застосунок. Система підтримує надсилання вкладень, автоматичне завантаження файлів, створення групових чатів та перегляд історії листування. Усі повідомлення пов’язані з конкретними користувачами та чатами, що забезпечує структуроване зберігання інформації та можливість подальшого аналізу даних.

Інтерфейс користувача розроблений для забезпечення інтуїтивної навігації та комфортної взаємодії із системою. Окремі модулі передбачені для роботи зі списком контактів, груповими чатами, профілем користувача, повідомленнями та вкладеннями. Графічний інтерфейс реалізований за допомогою бібліотеки Qt та підтримує сучасний дизайн із можливістю швидкого доступу до основних функцій месенджера.

З архітектурної точки зору система побудована за клієнт-серверною моделлю. Клієнтська частина відповідає за взаємодію з користувачем, відображення повідомлень та формування мережевих запитів. Серверна частина забезпечує обробку підключень, маршрутизацію повідомлень, роботу з базою даних PostgreSQL та підтримку TLS-з’єднання через OpenSSL. Для підвищення продуктивності сервер використовує механізми асинхронної обробки мережевих подій та connection pool для оптимізації роботи з базою даних.

Завдяки такій структурованій організації система обміну повідомленнями забезпечує ефективну та захищену онлайн-комунікацію між користувачами, підтримує передачу повідомлень і файлів у режимі реального часу, гарантує стабільність роботи при багатокористувацькому навантаженні та дозволяє реалізувати сучасний функціональний месенджер із високим рівнем безпеки та продуктивності.

Функціональні вимоги визначають основні можливості та поведінку програмної системи обміну повідомленнями, що забезпечують ефективну взаємодію користувачів у режимі реального часу. Розроблена система

повинна підтримувати стабільний обмін текстовими повідомленнями, безпечну автентифікацію користувачів та централізоване зберігання даних.

Функціональні вимоги:

- реєстрація користувачів із введенням логіна, імені користувача та пароля;
- автентифікація користувачів із перевіркою облікових даних та підтримкою refresh-токенів;
- встановлення захищеного TLS-з’єднання між клієнтом та сервером для безпечної передачі даних;
- обмін текстовими повідомленнями між користувачами у режимі реального часу;
- збереження історії повідомлень у базі даних PostgreSQL;
- автоматичне оновлення списку чатів та нових повідомлень без необхідності перезапуску застосунку;
- підтримка створення та управління приватними чатами між користувачами;
- відображення дати та часу надсилання повідомлень;
- підтримка системи авторизації та керування сесіями користувачів;
- збереження інформації про користувачів, повідомлення та токени авторизації у базі даних;
- обробка помилок з’єднання та повторне підключення клієнта до сервера;
- підтримка багатовіконного графічного інтерфейсу користувача у середовищі Qt;
- логування основних серверних подій, таких як підключення клієнтів, помилки авторизації та передача повідомлень;
- підтримка роботи кількох клієнтів одночасно за допомогою багатопотокової або асинхронної серверної обробки.

Нефункціональні вимоги визначають якісні характеристики системи, що впливають на її продуктивність, безпечність та надійність.

Нефункціональні вимоги:

- висока швидкість передачі повідомлень між клієнтами із мінімальною затримкою;
- забезпечення стабільної роботи сервера при одночасному підключенні декількох користувачів;
- захист даних користувачів за допомогою TLS-шифрування та безпечного зберігання паролів;
- надійність системи та стійкість до помилок мережевого з’єднання;
- масштабованість серверної архітектури для підтримки збільшення кількості користувачів;
- цілісність та узгодженість даних у базі PostgreSQL;
- зручність та інтуїтивність графічного інтерфейсу користувача;
- можливість подальшого розширення функціоналу системи без суттєвих змін архітектури;
- підтримка кросплатформеності клієнтського застосунку завдяки використанню Qt Framework;
- швидке відновлення роботи системи після технічних збоїв або втрати з’єднання.

1.5 Вимоги до інтерфейсу користувача

Інтерфейс користувача системи обміну повідомленнями повинен забезпечувати просту, зрозумілу та комфортну взаємодію користувача із програмним застосунком. Основною метою інтерфейсу є забезпечення швидкого доступу до функцій авторизації, перегляду чатів та обміну повідомленнями у режимі реального часу.

Головне вікно застосунку повинно містити список доступних чатів, область перегляду повідомлень та поле введення тексту повідомлення. Користувач повинен мати можливість швидко перемикатися між чатами, переглядати історію листування та надсилати нові повідомлення без затримок. Повідомлення повинні відображатися у зручному форматі із зазначенням часу надсилання та автора повідомлення.

Форма реєстрації повинна містити поля для введення логіна, імені користувача та пароля. Для підвищення зручності користувача передбачена можливість переходу між формами реєстрації та авторизації за допомогою інтерактивних посилань. Під час введення пароля система повинна підтримувати функцію приховування та відображення символів пароля.

Інтерфейс авторизації повинен забезпечувати зрозуміле інформування користувача про успішний вхід або виникнення помилок, наприклад неправильного логіна чи пароля. Усі повідомлення про помилки повинні бути короткими, зрозумілими та інформативними.

Важливою вимогою є підтримка адаптивності інтерфейсу та коректного масштабування елементів вікна при зміні розмірів застосунку. Графічний інтерфейс повинен мати сучасний вигляд, логічне розташування елементів та забезпечувати комфортне використання системи протягом тривалого часу.

Інтерфейс також повинен підтримувати оновлення повідомлень у реальному часі без необхідності ручного перезавантаження вікна. Для цього застосовуються механізми автоматичного оновлення списку повідомлень та чатів.

Особлива увага приділяється безпечності взаємодії користувача із системою. Під час встановлення з’єднання користувач повинен автоматично працювати через захищений TLS-канал без необхідності додаткових налаштувань. Усі критичні дії, пов’язані з авторизацією та передачею даних, повинні виконуватись із відповідним рівнем захисту.

Загалом інтерфейс користувача повинен забезпечувати швидку навігацію, мінімальну складність використання та ефективну взаємодію користувачів із системою обміну повідомленнями.

Висновки до розділу 1

У першому розділі проведено аналіз системи обміну повідомленнями як об'єкта дослідження. Визначено основні функціональні складові месенджера: реєстрація та автентифікація користувачів, обмін текстовими повідомленнями та файлами в режимі реального часу, підтримка приватних і групових чатів, а також відстеження статусу присутності учасників.

Здійснено UML-моделювання предметної області: побудовано діаграму прецедентів, що визначає сценарії взаємодії актора «Клієнт» із системою, діаграму послідовності для процесів авторизації та надсилання повідомлень, а також діаграму активності, що відображає алгоритм обробки повідомлення від введення користувачем до підтвердження доставки.

Проведено огляд існуючих рішень — месенджерів Telegram та Discord, виявлено їх переваги і недоліки, зокрема залежність від закритої серверної інфраструктури та обмежені можливості кастомізації, що підтверджує доцільність розробки власного програмного рішення.

На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до системи, а також вимоги до інтерфейсу користувача, що забезпечують зручну навігацію між чатами, оновлення контенту в реальному часі та захищену взаємодію через шифрований TLS-канал.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

ERwin — це професійний CASE-інструмент для моделювання та проєктування баз даних, який використовується для створення логічних і фізичних моделей даних, документування структури системи та візуалізації взаємозв’язків між сутностями. Основною перевагою ERwin є можливість побудови структурованої моделі бази даних, яка забезпечує цілісність даних, спрощує процес розробки та дозволяє ефективно керувати складними інформаційними системами. Інструмент широко застосовується під час розробки клієнт-серверних застосунків, корпоративних систем, систем електронного документообігу та месенджерів.

ERwin підтримує концептуальне, логічне та фізичне моделювання баз даних. Концептуальна модель дозволяє визначити основні сутності системи та зв’язки між ними без деталізації реалізації. Логічна модель містить атрибути, первинні та зовнішні ключі, типи зв’язків і структуру даних. Фізична модель дозволяє реалізувати структуру бази даних для конкретної системи управління базами даних із врахуванням типів даних, індексів, обмежень та SQL-коду.

Інструмент має графічний інтерфейс із підтримкою побудови ER-діаграм, автоматичної генерації SQL-структур, перевірки цілісності даних та документування проєкту. Використання ERwin значно полегшує аналіз архітектури системи, виявлення помилок у структурі даних та оптимізацію взаємодії між компонентами програмного забезпечення.

ER-діаграма (Entity-Relationship Diagram, діаграма «сутність-зв’язок») — це графічне представлення структури бази даних, яке відображає сутності предметної області, їх атрибути та логічні зв’язки між ними. ER-діаграми є

важливим етапом проєктування інформаційних систем, оскільки дозволяють наочно показати організацію даних та принципи їх взаємодії в межах системи.

Основними елементами ER-діаграми є сутності, атрибути та зв’язки. Сутності описують об’єкти предметної області, атрибути характеризують властивості цих об’єктів, а зв’язки визначають взаємодію між сутностями. Зв’язки можуть мати різну кардинальність: один-до-одного, один-до-багатьох або багато-до-багатьох. Завдяки ER-діаграмам можна забезпечити нормалізацію бази даних, уникнути дублювання інформації та підвищити ефективність роботи системи.

ER-діаграми широко використовуються під час розробки клієнт-серверних застосунків, оскільки дозволяють чітко визначити структуру зберігання даних, організувати взаємодію між таблицями та забезпечити масштабованість системи. Вони також слугують засобом документування проєкту та покращують взаєморозуміння між розробниками, аналітиками та адміністраторами баз даних. Логічна модель системи представлена на рисунку 2.1

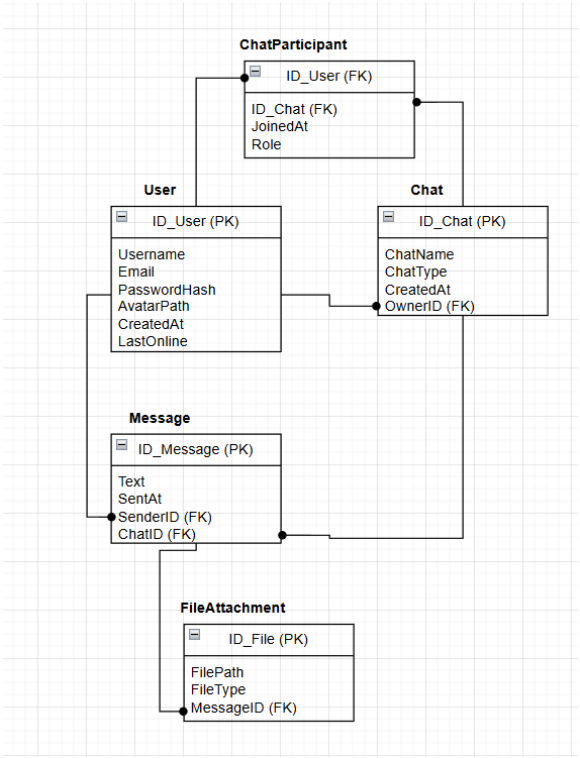

Рис. 2.1 Логічна модель системи “chat_db”

Дана ER-діаграма описує структуру бази даних клієнт-серверного месенджера, реалізованого з використанням технології Qt та SSL-з'єднання. Система підтримує обмін текстовими повідомленнями, надсилання файлів, створення групових чатів, управління профілями користувачів, статусами повідомлень та аватарами користувачів.

Центральною сутністю системи є таблиця Clients, яка зберігає інформацію про користувачів месенджера. Вона містить унікальний email користувача, ім'я користувача, хеш пароля, шлях до аватара, дату останньої активності та дату створення облікового запису. Кожен користувач може брати участь у кількох чатах, надсилати повідомлення та отримувати повідомлення від інших учасників системи.

Таблиця Chats призначена для зберігання інформації про чати. Вона містить тип чату (приватний або груповий), дату створення та користувача, який створив чат. Один чат може містити багатьох учасників та багато повідомлень.

Зв'язок між користувачами та чатами реалізований через проміжну таблицю ChatParticipants, яка забезпечує зв'язок типу «багато-до-багатьох». Таблиця містить ідентифікатор чату, ідентифікатор користувача, дату приєднання та роль користувача в чаті (адміністратор або учасник). Це дозволяє реалізувати функціонал групових чатів із розмежуванням прав доступу.

Таблиця Messages містить інформацію про всі повідомлення в системі. Для кожного повідомлення зберігається ідентифікатор чату, відправник, текст повідомлення, тип повідомлення (текст або файл), дата створення, а також ознаки редагування та видалення повідомлення. Система підтримує функції редагування повідомлень і логічного видалення без фізичного видалення запису з бази даних.

Для підтримки передачі файлів використовується таблиця Files, яка пов'язана з таблицею повідомлень через зовнішній ключ message_id. Вона

містить назву файлу, шлях до файлу на сервері, тип файлу, його розмір та дату завантаження. Така структура дозволяє окремо зберігати метадані файлів і забезпечує підтримку надсилання документів, зображень, архівів та інших вкладень у чаті.

Контроль статусу прочитання повідомлень реалізовано через таблицю MessageStatus. У ній зберігається інформація про те, чи було повідомлення прочитане певним користувачем, а також час прочитання. Це дозволяє реалізувати функціонал відображення статусів доставки та перегляду повідомлень у реальному часі.

Представлена ER-діаграма демонструє логічно структуровану архітектуру бази даних месенджера, яка забезпечує підтримку приватних та групових чатів, передачу файлів, управління користувачами та контроль стану повідомлень. Використання такої структури дозволяє масштабувати систему, додавати нові функції та підтримувати ефективну взаємодію між клієнтською та серверною частинами застосунку.

2.2 Вибір системи управління базами даних

При розробці клієнт-серверного месенджера одним із найважливіших етапів є правильне проєктування структури бази даних. База даних виступає центральним компонентом системи, який забезпечує збереження інформації про користувачів, чати, повідомлення, вкладені файли та взаємодію між учасниками чатів. Від коректної організації структури даних залежить стабільність роботи застосунку, швидкість обробки повідомлень, підтримка групових чатів та можливість масштабування системи в майбутньому.

Для реалізації проєкту було використано реляційну систему управління базами даних MySQL. Вибір саме реляційної СУБД обумовлений необхідністю роботи зі структурованими даними та великою кількістю взаємозв’язків між сутностями системи. MySQL забезпечує підтримку SQL-запитів, зовнішніх ключів, транзакцій, індексів та механізмів контролю

цілісності даних, що є критично важливим для роботи чат-системи в режимі реального часу.

При виборі СУБД були враховані такі критерії:

- підтримка реляційної моделі даних;
- можливість роботи зі складними SQL-запитами;
- швидкість обробки великої кількості повідомлень;
- підтримка транзакцій та цілісності даних;
- простота інтеграції з серверною частиною застосунку на C++ та Qt;
- наявність документації та широкої спільноти розробників;
- підтримка масштабування та оптимізації продуктивності.

На основі проведеного аналізу було обрано MySQL Workbench як інструмент для проєктування структури бази даних та створення ER-діаграми системи. MySQL Workbench дозволяє візуально моделювати структуру таблиць, визначати зв’язки між сутностями, створювати первинні та зовнішні ключі, а також автоматично генерувати SQL-код для створення бази даних.

ER-діаграма (Entity-Relationship Diagram, діаграма «сутність-зв’язок») є одним із ключових інструментів логічного моделювання баз даних. Вона дозволяє наочно представити структуру системи, основні сутності, їх атрибути та взаємозв’язки між ними. Використання ER-діаграми значно спрощує процес розробки, допомагає уникнути дублювання даних та забезпечує правильну організацію структури бази даних ще на етапі проєктування.

Основними компонентами ER-діаграми є:

- сутності — об’єкти предметної області;
- атрибути — характеристики сутностей;
- зв’язки — логічні взаємозв’язки між сутностями;
- кардинальність — тип зв’язку між таблицями («один-до-одного», «один-до-багатьох», «багато-до-багатьох»).

Логічна модель системи представлена на рисунку 2.2

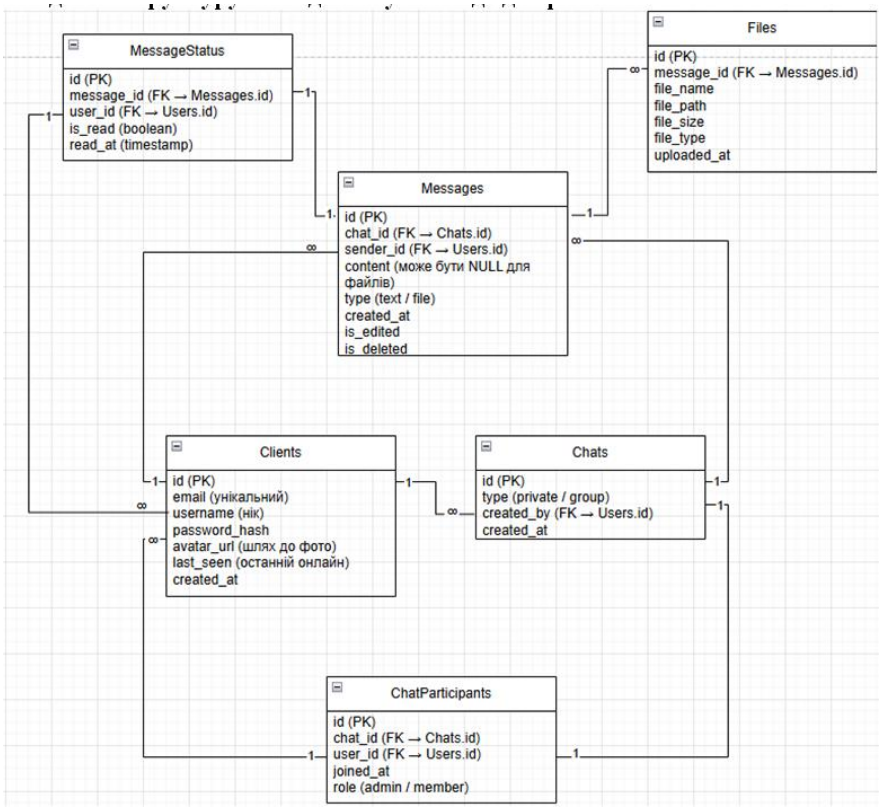

Рис. 2.2 ER-діаграма бази даних месенджера

Структура бази даних складається з кількох основних таблиць, які забезпечують роботу чат-системи.

Таблиця User зберігає інформацію про користувачів системи. У ній містяться унікальний ідентифікатор користувача, ім’я користувача, електронна пошта, хеш пароля, шлях до аватару користувача, дата створення облікового запису та інформація про останню активність користувача в системі. Кожен користувач може брати участь у кількох чатах та надсилати повідомлення.

Таблиця Chat містить інформацію про чати. Вона включає ідентифікатор чату, назву чату, тип чату (приватний або груповий), дату створення та ідентифікатор власника чату. Наявність поля ChatType дозволяє системі підтримувати як приватні діалоги між двома користувачами, так і групові чати з декількома учасниками.

Таблиця ChatParticipant реалізує зв’язок типу «багато-до-багатьох» між користувачами та чатами. Один користувач може бути учасником багатьох чатів, а один чат може містити багатьох користувачів. У таблиці також зберігається дата приєднання користувача до чату та його роль у групі (наприклад, admin або member). Це дозволяє реалізувати механізм адміністрування групових чатів.

Таблиця Message відповідає за збереження повідомлень користувачів. Вона містить текст повідомлення, час надсилання, ідентифікатор відправника та ідентифікатор чату, до якого належить повідомлення. Кожне повідомлення пов’язане з конкретним користувачем та конкретним чатом через зовнішні ключі.

У системі також реалізована підтримка передачі файлів. Для цього використовується таблиця FileAttachment, яка містить інформацію про вкладення: шлях до файлу, тип файлу та ідентифікатор повідомлення, до якого прикріплений файл. Це дозволяє надсилати зображення, документи, архіви, відео та інші типи файлів безпосередньо в чаті.

Крім основних таблиць, у повній моделі бази даних також реалізовані додаткові механізми:

- таблиця MessageStatus для відстеження статусу прочитання повідомлень;
- підтримка редагування повідомлень через поля is_edited;
- підтримка логічного видалення повідомлень через поле is_deleted;
- збереження типу повідомлення (text/file);
- підтримка аватарів користувачів;
- збереження часу останньої активності користувача.

Таблиця MessageStatus дозволяє реалізувати механізм прочитаних повідомлень. Вона містить посилання на повідомлення та користувача, статус прочитання і час, коли повідомлення було прочитане. Такий підхід дозволяє коректно працювати як із приватними, так і з груповими чатами.

Таким чином структура бази даних побудована за принципами нормалізації та реляційного моделювання. Основні зв’язки між таблицями мають тип:

- один користувач → багато повідомлень;
- один чат → багато повідомлень;
- один чат → багато учасників;
- один користувач → багато чатів;
- одне повідомлення → багато вкладень.

Подібна структура забезпечує:

- цілісність даних;
- швидкий доступ до повідомлень;
- можливість масштабування системи;
- підтримку групових чатів;
- підтримку передачі файлів;
- реалізацію механізмів редагування та видалення повідомлень;
- контроль статусів прочитання повідомлень.

Після завершення проєктування база даних була реалізована у середовищі MySQL Workbench. Було створено всі таблиці, налаштовано зовнішні ключі, індекси та зв’язки між сутностями. Це дозволило забезпечити стабільну роботу серверної частини застосунку та ефективну взаємодію клієнта із сервером у режимі реального часу.

Таким чином, розроблена структура бази даних створює надійну основу для функціонування чат-системи, забезпечуючи ефективне управління користувачами, повідомленнями, файлами та груповими чатами, а також можливість подальшого розширення функціональності програмного забезпечення.

2.3 Розробка структури інформаційної бази

У процесі створення системи обміну повідомленнями особливу увагу приділено розробці структури інформаційної бази, оскільки саме вона є основою збереження, обробки та взаємодії з даними. Ретельно спроектована структура бази даних забезпечує логічну цілісність, ефективність виконання запитів та можливість масштабування у майбутньому.

Для підтримки повноцінного функціонування системи структура бази даних містить основні таблиці з обмеженнями цілісності та індекси для прискорення виконання найбільш навантажених запитів. Такий підхід дозволяє забезпечити контроль над цілісністю інформації на рівні СУБД та мінімізувати час відгуку при роботі з великими обсягами повідомлень. У наступних підрозділах описано реалізацію кожного з цих компонентів.

2.3.1 Створення бази даних. Як систему управління базами даних обрано PostgreSQL — надійну реляційну СУБД з відкритим вихідним кодом, що підтримує транзакції, складні запити, зовнішні ключі та широкий набір типів даних. Для взаємодії серверного C++ коду з PostgreSQL використовується бібліотека libpqxx, яка надає типобезпечний інтерфейс для виконання параметризованих SQL-запитів. База даних розгортається на Linux-сервері під іменем chat_db з виділеним користувачем chat_user, якому надаються необхідні привілеї на роботу з таблицями.

2.3.2 Створення таблиць. Процес створення таблиць є ключовим етапом у розробці структури бази даних, оскільки кожна таблиця відображає окрему сутність системи і містить стовпці з відповідними атрибутами. Для забезпечення ефективного зберігання даних визначено правильну структуру таблиць відповідно до вимог бізнес-логіки застосунку. Усі таблиці створюються за допомогою команди CREATE TABLE з явним визначенням типів даних, обмежень первинних і зовнішніх ключів, а також перевірочних обмежень CHECK.

Загалом у базі даних реалізовано шість основних таблиць:

users — зберігає облікові записи користувачів: електронну адресу (унікальну), нікнейм, хеш пароля, шлях до аватара, код і термін дії коду скидання пароля, час останньої активності та дату реєстрації.

chats — містить записи про чати з розмежуванням типу (private або group), назву групи для групових чатів, ідентифікатор користувача-творця та час створення.

chat_participants — таблиця учасників чату, що реалізує зв'язок «багато до багатьох» між таблицями users та chats. Містить роль учасника (admin або member) та час приєднання. Унікальний складений ключ (chat_id, user_id) унеможливлює дублювання учасників у межах одного чату.

messages — зберігає повідомлення з прив'язкою до чату та відправника, текстовий вміст, тип повідомлення (text або file), час відправлення, а також прапорці редагування та м'якого видалення.

files — містить метадані прикріплених файлів: оригінальну назву, шлях зберігання на сервері, розмір, MIME-тип та час завантаження. Пов'язана з таблицею messages через зовнішній ключ з каскадним видаленням.

message_status — таблиця статусів прочитання повідомлень, що фіксує факт прочитання кожного повідомлення кожним учасником чату. Унікальний складений ключ (message_id, user_id) гарантує одиничність запису статусу для кожної пари.

Кожна таблиця має первинний ключ типу SERIAL, що забезпечує автоматичну генерацію унікальних ідентифікаторів. Для встановлення взаємозв'язків між таблицями використовуються зовнішні ключі (FOREIGN KEY). Наприклад, поле chat_id у таблицях chat_participants, messages та message_status пов'язане з первинним ключем таблиці chats, а поле user_id — з первинним ключем таблиці users. При видаленні батьківського запису для більшості зовнішніх ключів застосовується стратегія ON DELETE CASCADE, що автоматично видаляє залежні записи, а для поля sender_id у таблиці messages — ON DELETE SET NULL, що дозволяє зберегти

повідомлення навіть після видалення облікового запису відправника.

Для перевірки коректності значень використовуються обмеження CHECK: зокрема, поле type таблиці chats приймає лише значення 'private' або 'group', поле type таблиці messages — лише 'text' або 'file', а поле role таблиці chat_participants — лише 'admin' або 'member'.

Для коректного зберігання даних використовуються такі типи:

SERIAL — для автоматично генерованих первинних ключів;

VARCHAR(n) — для рядків обмеженої довжини, зокрема електронних адрес, нікнеймів, типів і назв;

TEXT — для рядків необмеженої довжини, зокрема хешів паролів, шляхів до файлів та вмісту повідомлень;

TIMESTAMP — для зберігання дати і часу подій;

BOOLEAN — для прапорців редагування, видалення та прочитання;

INTEGER — для числових значень, зокрема розміру файлів і зовнішніх ключів.

2.3.3 Створення індексів. Для підвищення продуктивності виконання запитів, що найчастіше використовуються при роботі з повідомленнями, у базі даних створено п'ять індексів. Індекс idx_messages_chat_id на полі chat_id таблиці messages прискорює завантаження історії повідомлень конкретного чату. Індекс idx_messages_sender_id на полі sender_id оптимізує пошук повідомлень за відправником. Індекс idx_files_message_id на полі message_id таблиці files прискорює отримання файлів, прив'язаних до конкретного повідомлення. Індекси idx_status_message_id та idx_status_user_id на відповідних полях таблиці message_status забезпечують швидку перевірку статусу прочитання. Застосування індексів суттєво знижує час виконання запитів на вибірку при зростанні обсягу даних.

2.4 Схема та фізична структура бази даних

Фізична структура бази даних є важливим елементом проєктування,

який визначає зберігання, організацію та взаємозв'язки даних. У межах даної системи таблиці бази даних створювалися за допомогою SQL-запитів мови PostgreSQL. Це дозволило детально налаштувати атрибути таблиць, встановити обмеження цілісності даних, включаючи первинні та зовнішні ключі з каскадними діями, а також задати типи даних відповідно до специфіки збережуваної інформації.

На схемі бази даних наочно зображено всі таблиці, поля, типи даних, а також взаємозв'язки між ними. Така схема дозволяє краще зрозуміти логіку структури, зручно аналізувати зв'язки між сутностями та визначити залежності, необхідні для ефективного функціонування системи.

Отже, фізична структура бази даних є логічно впорядкованою, гнучкою та масштабованою. Вона враховує специфіку комунікаційних даних, забезпечує надійне зберігання інформації та підтримує важливі механізми забезпечення цілісності через зовнішні ключі, унікальні обмеження та перевірочні обмеження на рівні СУБД. Такий підхід дозволяє легко розширювати систему у майбутньому, додаючи нові сутності або функціональність, без шкоди для стабільності або якості даних.

Висновки до розділу 2

У другому розділі спроєктовано інформаційне забезпечення системи обміну повідомленнями. Розроблено логічну модель даних у вигляді ER-діаграми, що охоплює шість основних сутностей: users, chats, chat_participants, messages, files та message_status, із чітко визначеними зв'язками типу «один-до-багатьох» та «багато-до-багатьох» між ними.

Обґрунтовано вибір реляційної СУБД PostgreSQL як системи зберігання даних завдяки підтримці транзакцій, зовнішніх ключів, складних SQL-запитів та можливостей масштабування. Для взаємодії серверного C++ коду з базою даних обрано бібліотеку libpqxx, що надає типобезпечний інтерфейс параметризованих запитів.

Розроблено фізичну структуру бази даних: визначено типи даних стовпців, первинні та зовнішні ключі з каскадними стратегіями ON DELETE CASCADE та ON DELETE SET NULL, перевірочні обмеження CHECK на значення полів type та role, а також унікальні складені ключі для запобігання дублюванню записів. Для підвищення продуктивності найбільш навантажених вибіркових запитів створено п'ять індексів на ключових полях таблиць messages та message_status.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Абстракції предметної області

Абстракції предметної області в UML використовуються для Абстракції предметної області в UML використовуються для узагальненого опису основних сутностей системи, їх характеристик та взаємозв’язків. Вони дозволяють сформувати цілісне уявлення про структуру програмного забезпечення та визначити ключові компоненти, які беруть участь у роботі системи. Використання абстракцій дає можливість спростити аналіз складних програмних рішень, виділивши лише найважливіші елементи предметної області без деталізації технічної реалізації.

Основною перевагою абстракцій є можливість структурування системи на окремі логічні компоненти, кожен з яких виконує власну роль та має визначені властивості й обов’язки. Це дозволяє забезпечити зрозумілість архітектури системи, полегшити процес моделювання та підготувати основу для подальшого проєктування UML-діаграм, таких як діаграми класів, діяльності та послідовності. Абстракції також допомагають визначити основні сценарії використання системи та взаємодію між її компонентами.

У розробленому месенджері абстракції предметної області описують основні сутності, необхідні для забезпечення обміну повідомленнями між користувачами, організації чатів, керування сесіями та взаємодії із серверною частиною системи. Кожна абстракція містить набір важливих властивостей та функцій, які характеризують її призначення у системі.

Абстракція «Користувач» описує зареєстрованого користувача месенджера. Вона містить такі властивості, як ім’я користувача, email, пароль, статус активності та роль користувача. Основними обов’язками цієї абстракції є реєстрація у системі, авторизація, надсилання та отримання повідомлень, а також редагування власного профілю. Дана абстракція є центральною складовою системи, оскільки саме користувач ініціює більшість

процесів у месенджері.

Абстракція «Повідомлення» представляє процес обміну текстовими повідомленнями між користувачами. Вона включає текст повідомлення, час відправлення, інформацію про відправника та отримувача, а також статус доставки. Основними функціями цієї абстракції є збереження тексту повідомлення, його передача між користувачами, відображення статусу доставки та можливість видалення повідомлення.

Абстракція «Чат» описує середовище взаємодії користувачів. Вона містить назву чату, список учасників, тип чату та дату його створення. Її обов’язками є створення чатів, додавання або видалення учасників, а також збереження історії повідомлень. У системі підтримуються як приватні, так і групові чати.

Абстракція «Сервер» відповідає за обробку клієнтських запитів та забезпечення роботи мережевої взаємодії. До її властивостей належать адреса сервера, порт, стан роботи та кількість активних підключень. Основні функції сервера полягають в обробці запитів клієнтів, передачі повідомлень між користувачами, виконанні автентифікації та керуванні активними сесіями.

Абстракція «Сесія» використовується для контролю авторизованих підключень користувачів. Вона містить session ID, час створення та завершення сесії, а також її поточний стан. До обов’язків сесії належать ідентифікація користувача, збереження стану авторизації, контроль активності та завершення сеансу роботи.

Абстракція «База даних» забезпечує збереження інформації та доступ до неї. Вона включає тип системи управління базами даних, обсяг даних та стан підключення. Основними функціями є зберігання користувачів і повідомлень, виконання SQL-запитів та забезпечення доступу до інформації.

Таким чином, виділені абстракції формують основу предметної області розробленого месенджера. Вони відображають ключові компоненти системи та їх взаємодію між собою. Абстракції «Користувач»,

«Повідомлення» і «Чат» відповідають за безпосередню взаємодію користувачів, тоді як «Сервер», «Сесія» та «База даних» забезпечують стабільне функціонування системи та обробку інформації. Використання таких абстракцій дозволяє зробити архітектуру застосунку більш структурованою, зрозумілою та придатною для подальшого масштабування.

Рис. 3.1 – Абстракції предметної області

3.2 Моделювання структури та взаємодії об'єктів

Наступним кроком є моделювання структури та взаємодії об'єктів, що дозволяє чітко визначити складові системи та їхні взаємозв'язки. На цьому етапі важливо не лише відобразити логіку функціонування системи обміну повідомленнями, а й чітко структурувати її складові, визначити, які об'єкти існують, які зв'язки між ними встановлюються, та як вони взаємодіють у процесі виконання функціоналу. Моделювання структури допомагає візуалізувати загальну архітектуру системи та слугує основою для подальшого її програмного втілення.

Для моделювання застосовується стандартна мова UML, що є загальноприйнятим інструментом для побудови об'єктно-орієнтованих моделей. Особливу роль на цьому етапі відіграють діаграми, що відображають як статичні, так і динамічні аспекти системи. Статичні аспекти представлені діаграмою класів, яка формує основу логічної структури системи та відображає сутності користувача, чату, повідомлення і файлу разом із їхніми атрибутами та взаємозв'язками. Динамічні аспекти ілюструються через діаграму кооперацій, що показує взаємодію об'єктів у процесі виконання конкретних сценаріїв, таких як надсилання повідомлення, створення групового чату або відновлення паролю [9].

Діаграма класів була побудована ще на етапі аналізу, оскільки саме вона дозволяє сформувати уявлення про функціональну модель системи та спроектувати об'єкти, що реалізуватимуть ключові можливості майбутнього програмного продукту. На основі цієї діаграми згодом формуються програмні класи клієнтської та серверної частин із відповідними атрибутами та методами, а також реалізуються міжоб'єктні взаємозв'язки, зокрема між користувачами, чатами та повідомленнями.

Діаграма класів (Class Diagram) є одним із основних типів UML-діаграм, що використовується для відображення статичної структури програмної системи. Вона дозволяє описати основні класи системи, їх атрибути, методи та взаємозв’язки між ними. За допомогою діаграми класів можна сформувати загальне уявлення про архітектуру програмного забезпечення, логіку взаємодії об’єктів та структуру даних, які використовуються у системі. Це дає змогу ще на етапі проєктування визначити основні компоненти системи та їх функціональне призначення.

Кожен клас у UML-діаграмі складається з трьох частин: назви класу, атрибутів та методів. Атрибути описують характеристики об’єкта, а методи визначають його поведінку та доступні операції. Важливу роль також відіграють зв’язки між класами, які показують, яким чином об’єкти взаємодіють між собою. Це дозволяє відобразити структуру системи на

високому рівні абстракції та забезпечити зрозумілість моделі для всіх учасників розробки [9].

Діаграма класів є основою для подальшої реалізації програмного забезпечення, оскільки саме на її базі створюється структура бази даних, логіка взаємодії компонентів та архітектура застосунку. Вона також допомагає уникнути дублювання функціональності, забезпечує кращу організацію коду та полегшує підтримку і масштабування системи. Крім того, діаграма класів виконує роль документації, що значно спрощує розуміння структури системи новими розробниками.

Під час створення діаграми класів важливо правильно визначити ключові сутності предметної області та їх взаємозв'язки. Це дозволяє сформувати логічну модель системи, яка буде максимально наближена до реальних процесів. Використання UML-діаграм класів забезпечує ефективне планування архітектури програмного продукту та сприяє підвищенню якості його реалізації.

Розроблена діаграма класів представлена на рис. 3.2

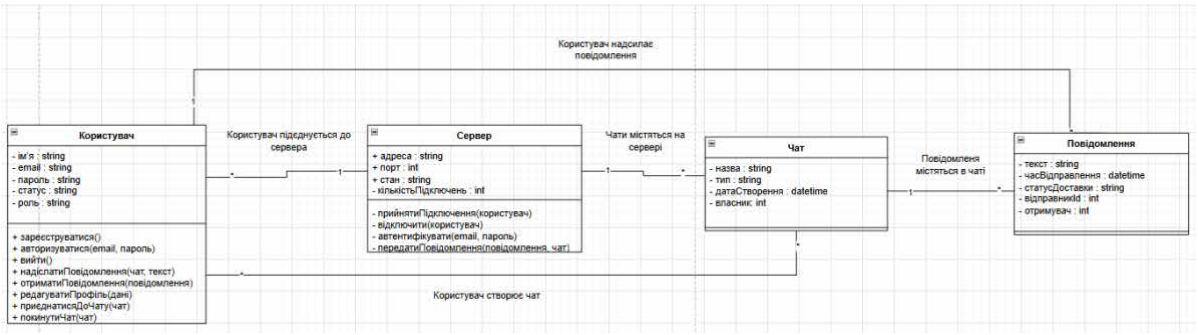

Рис. 3.2 – Діаграма класів

Дана діаграма класів описує структуру клієнт-серверного чат-застосунку та демонструє взаємодію між основними компонентами системи: користувачем, сервером, чатом і повідомленнями. Вона відображає основні сутності системи, їх атрибути, методи та зв'язки між ними.

Центральним елементом діаграми є клас Користувач. Він містить основні атрибути користувача: ім'я, email, пароль, статус та роль. Також клас

включає методи для реєстрації, авторизації, виходу із системи, надсилання та отримання повідомлень, редагування профілю, а також приєднання та виходу з чату. Це дозволяє реалізувати повний цикл взаємодії користувача із системою.

Клас Сервер відповідає за обробку клієнтських підключень та передачу повідомлень між користувачами. Він містить атрибути адреси сервера, порту, стану роботи та кількості активних підключень. Основними методами сервера є прийняття та відключення користувачів, автентифікація та передача повідомлень у чатах. Сервер виступає центральним компонентом системи, який координує взаємодію між усіма іншими об’єктами.

Клас Чат описує окремий чат у системі. Він містить назву чату, тип (приватний або груповий), дату створення та інформацію про власника. Чат використовується для об’єднання користувачів та збереження повідомлень. Завдяки цьому забезпечується підтримка як особистих, так і групових чатів.

Клас Повідомлення відповідає за роботу з повідомленнями, які передаються між користувачами. Він містить текст повідомлення, дату відправлення, статус доставки, а також ідентифікатори відправника та отримувача. Це дозволяє реалізувати функціонал надсилання, отримання та збереження історії листування.

На діаграмі також показані зв’язки між класами. Користувач підключається до сервера, сервер містить чати, а кожен чат включає набір повідомлень. Крім того, користувачі можуть створювати чати та надсилати повідомлення через систему. Такі зв’язки демонструють логіку взаємодії компонентів і відображають реальну структуру роботи чат-застосунку.

Таким чином, діаграма класів демонструє цілісну архітектуру системи обміну повідомленнями. Вона показує, як організована взаємодія між користувачами, сервером та чатами, а також дозволяє оцінити структуру програмного забезпечення ще до етапу реалізації. Це забезпечує кращу зрозумілість системи, спрощує її подальшу розробку та створює основу для масштабування функціоналу застосунку.

3.3 Архітектура програмного забезпечення

Існує декілька основних підходів до побудови архітектури програмного забезпечення, кожен із яких визначає спосіб взаємодії компонентів системи та організацію обробки даних. Найпростішим підходом є монолітна архітектура, у якій усі модулі системи реалізовані як єдиний програмний застосунок. Такий підхід є зручним на початкових етапах розробки, однак зі збільшенням функціональності система стає складною в підтримці та масштабуванні.

Багаторівнева архітектура (multi-tier architecture) передбачає розділення системи на окремі логічні рівні, кожен із яких відповідає за власний набір функцій. Це дозволяє підвищити модульність, спростити тестування та забезпечити незалежний розвиток окремих компонентів системи. Подібний підхід широко використовується у сучасних мережевих застосунках, системах електронної комерції, веб-сервісах та чат-платформах.

Існують також сервісно-орієнтовані архітектури (SOA), де функціональність системи поділена на окремі сервіси, які взаємодіють між собою через стандартизовані інтерфейси. Розвитком цього підходу є мікросервісна архітектура, у якій кожен сервіс виконує одну конкретну задачу та може масштабуватися незалежно від інших компонентів. Також використовуються подієво-орієнтовані архітектури, де взаємодія між компонентами здійснюється через події та асинхронний обмін повідомленнями.

Для реалізації чат-системи було обрано клієнт-серверну багаторівневу архітектуру, оскільки вона забезпечує чітке розділення функцій між клієнтською частиною, сервером застосунку та системою збереження даних. Такий підхід дозволяє централізовано обробляти повідомлення, підтримувати багатокористувацьку взаємодію, забезпечувати контроль доступу та ефективно працювати з базою даних.

Основною перевагою клієнт-серверної архітектури є розділення ролей між компонентами системи:

- клієнт відповідає за взаємодію з користувачем;
- сервер виконує обробку запитів та реалізацію бізнес-логіки;
- база даних забезпечує централізоване збереження інформації.

У процесі розробки месенджера було реалізовано трирівневу клієнт-серверну архітектуру, яка складається з:

- клієнтського застосунку;
- сервера застосунку;
- сервера бази даних.

Архітектура системи представлена на рисунку 3.3

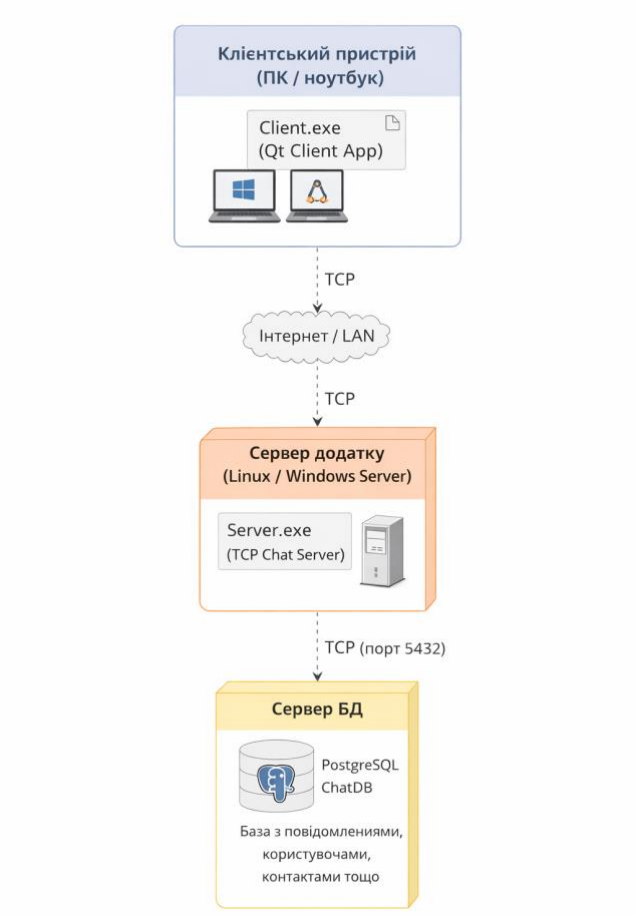

Рис. 3.3 Архітектура клієнт-серверного месенджера

На верхньому рівні знаходиться клієнтський застосунок, реалізований за допомогою фреймворку Qt мовою програмування C++. Клієнтська частина представлена програмою Client.exe, яка забезпечує графічний інтерфейс користувача та взаємодію із сервером. Саме через клієнтський застосунок користувач може:

- авторизуватися в системі;
- переглядати список контактів;
- створювати приватні та групові чати;
- надсилати текстові повідомлення;
- передавати файли;
- редагувати та видаляти повідомлення;
- змінювати дані профілю;
- переглядати статуси користувачів.

Інтерфейс користувача побудований із використанням компонентів Qt Widgets, що забезпечує кросплатформеність застосунку та зручну роботу як у Windows, так і в Linux-середовищі. Клієнт взаємодіє із сервером через TCP-з’єднання з використанням класу QSslSocket, що дозволяє забезпечити захищену передачу даних мережею.

Другий рівень архітектури представлений сервером застосунку. Серверна частина реалізована у вигляді окремого TCP-сервера, який приймає підключення від клієнтів, обробляє JSON-запити та координує обмін повідомленнями між користувачами. Сервер виконує основну бізнес-логіку системи, а саме:

- автентифікацію та реєстрацію користувачів;
- створення чатів;
- управління групами;
- обробку повідомлень;
- передачу файлів;
- синхронізацію історії повідомлень;

- обробку статусів користувачів;
- роботу з аватарами;
- контроль доступу до чатів.

Для обміну даними між клієнтом і сервером використовується формат JSON, що дозволяє легко структурувати повідомлення та забезпечує просту інтеграцію між компонентами системи. Передача даних здійснюється через TCP-протокол, який гарантує надійність доставки повідомлень та правильний порядок їх отримання.

На третьому рівні знаходиться сервер бази даних PostgreSQL, який забезпечує централізоване збереження всієї інформації системи. У базі даних зберігаються:

- користувачі;
- контакти;
- чати;
- учасники груп;
- повідомлення;
- вкладені файли;
- статуси повідомлень;
- інформація про аватари;
- часові мітки активності користувачів.

Використання PostgreSQL забезпечує:

- підтримку реляційної моделі даних;
- високу надійність роботи;
- підтримку транзакцій;
- механізми забезпечення цілісності даних;
- ефективну роботу з великою кількістю повідомлень;
- підтримку складних SQL-запитів;
- масштабованість системи.

Сервер застосунку взаємодіє з PostgreSQL через окремий рівень доступу до даних, який виконує SQL-запити, отримує результати та передає їх бізнес-логіці сервера. Подібне розділення дозволяє ізолювати логіку роботи з базою даних від інших компонентів системи та спрощує подальшу модифікацію структури БД.

Архітектура системи побудована за принципом чіткого розділення відповідальностей:

- клієнт відповідає за інтерфейс користувача;
- сервер реалізує бізнес-логіку;
- база даних забезпечує збереження інформації.

Такий підхід забезпечує:

- масштабованість системи;
- спрощення підтримки програмного коду;
- можливість незалежного оновлення компонентів;
- підвищення безпеки;
- централізоване управління даними;
- ефективну синхронізацію повідомлень між користувачами;
- можливість подальшого розширення функціональності.

Крім того, використання клієнт-серверної архітектури дозволяє реалізувати підтримку багатокористувацького режиму роботи, забезпечити контроль одночасних підключень та організувати стабільний обмін повідомленнями у режимі реального часу.

Таким чином, обрана архітектура забезпечує надійну, гнучку та масштабовану основу для функціонування чат-системи, дозволяючи ефективно організувати взаємодію між користувачами, сервером застосунку та базою даних.

3.4 Організаційна структура програмного забезпечення

3.4.1 Діаграма пакетів. Діаграма пакетів у UML є інструментом для візуалізації високорівневої організації програмного забезпечення та відображення логічних залежностей між різними частинами системи. Пакет у такій діаграмі – це групування класів, інтерфейсів або інших елементів моделі, яке дозволяє структурувати програму на окремі логічні компоненти та приховати деталі реалізації всередині пакетів.

Основна мета діаграми пакетів – продемонструвати структуру системи на рівні модулів і взаємозв'язки між ними, що особливо корисно для великих проєктів, де багато класів та компонентів. Вона показує, які частини системи залежать одна від одної, і допомагає визначити точки інтеграції, а також спростити підтримку та масштабування програмного забезпечення [9].

У діаграмі пакетів зв'язки між пакетами можуть бути різних типів: залежності, імпорти, реалізації або асоціації. Це дозволяє розробникам швидко оцінити, які модулі впливають на інші, і визначити області, які можна модифікувати без ризику порушення роботи системи. Крім того, діаграма пакетів допомагає визначити логічні блоки для командної роботи, оскільки кожен пакет можна призначити конкретній групі розробників.

Діаграми пакетів використовуються як для проєктування нових систем, так і для документування існуючих, забезпечуючи узгоджене уявлення про структуру програмного забезпечення на високому рівні. Вони добре доповнюють діаграми класів, оскільки дозволяють бачити не окремі класи, а їх логічні об'єднання, і тим самим спрощують управління складними проєктами.

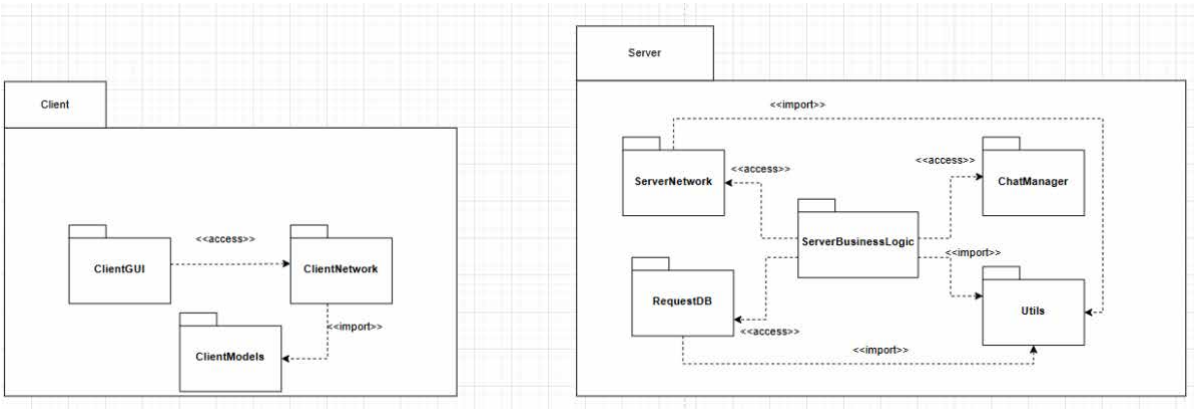

Рис. 3.4 Діаграма пакетів

Дана діаграма пакетів відображає архітектуру системи обміну повідомленнями, яка складається з двох самостійних підсистем: клієнтської частини (Client) і серверної частини (Server), кожна з яких внутрішньо поділена на функціональні пакети.

Клієнтська підсистема містить три пакети. Пакет ClientGUI реалізує увесь графічний інтерфейс користувача: вікна авторизації та реєстрації, головне вікно чату, діалоги профілю, відновлення пароля та створення групових чатів. Через зв'язок <<access>> він звертається до пакету ClientNetwork, який відповідає за мережеву взаємодію з сервером через захищене TLS-з'єднання. Пакет ClientModels містить структури даних, якими оперує клієнтська частина, і пов'язаний з ClientNetwork через зв'язок <<import>>, забезпечуючи мережевому рівню доступ до необхідних моделей.

Серверна підсистема організована у п'ять пакетів. Вхідною точкою є пакет ServerNetwork, який реалізує асинхронне приймання з'єднань через механізм epoll та управляє TLS-рукостисканнями. Через зв'язок <<access>> він передає оброблені повідомлення до пакету ServerBusinessLogic, що містить логіку маршрутизації запитів і координує роботу решти компонентів. Пакет RequestDB забезпечує роботу з базою даних PostgreSQL через пул з'єднань і пов'язаний з ServerBusinessLogic зв'язком <<access>>. Пакет ChatManager реалізує сервіси чатів: завантаження повідомлень, створення приватних і групових чатів, управління учасниками — і взаємодіє з

ServerBusinessLogic через зв'язок <<access>>. Пакет Utils є спільною бібліотекою утиліт, яка надає функції кодування Base64, хешування паролів, генерації унікальних імен файлів та відправки електронних листів; до нього звертаються через <<import>> пакети ServerBusinessLogic, ChatManager та ServerNetwork.

Таким чином, архітектура побудована за принципом чіткого розділення відповідальності: мережевий рівень ізольований від бізнес-логіки, бізнес-логіка ізольована від бази даних, а допоміжні утиліти виокремлені в незалежний пакет. Це забезпечує зручність супроводу, можливість незалежного тестування кожного компонента та легкість масштабування системи.

3.5 Вибір інструментарію для створення програмного забезпечення

Для розробки програмного забезпечення було здійснено ретельний вибір інструментів та технологій, що дозволяють створити сучасну, надійну та безпечну систему обміну повідомленнями в реальному часі. Основними критеріями при виборі були: продуктивність, безпека передачі даних, кросплатформеність клієнтської частини, наявність документації та активної спільноти, а також можливість ефективної роботи з великою кількістю одночасних підключень.

Клієнтська частина додатку реалізована з використанням фреймворку Qt 6, що надає повний набір засобів для побудови графічного інтерфейсу користувача. Qt забезпечує кросплатформеність, компонентну архітектуру на основі сигналів і слотів, вбудовану підтримку мережевої взаємодії через QSslSocket, а також зручні засоби для роботи з JSON, зображеннями та анімацією. Мова програмування C++ обрана для обох частин системи завдяки своїй високій продуктивності, детальному контролю над пам'яттю та широкому спектру стандартних і сторонніх бібліотек.

Серверна частина реалізована як автономний C++ процес під управлінням операційної системи Linux. Для обробки великої кількості одночасних підключень без надмірного навантаження використовується механізм epoll — стандартний Linux-інтерфейс для асинхронного введення-виведення, що дозволяє ефективно моніторити події на великій кількості сокетів в єдиному потоці виконання.

Захист мережевої взаємодії між клієнтом і сервером забезпечується бібліотекою OpenSSL, яка реалізує протокол TLS. Усі дані передаються у зашифрованому вигляді, що унеможливлює їх перехоплення. Бібліотека використовується як на стороні сервера для прийому захищених з'єднань, так і на стороні клієнта через абстракцію QSslSocket у Qt, а також для реалізації SMTPS-клієнта, що надсилає коди скидання пароля електронною поштою.

Для зберігання та обробки даних обрана реляційна система управління базами даних PostgreSQL, яка забезпечує надійне збереження інформації, підтримку транзакцій, складних запитів та узгодженість даних. PostgreSQL дозволяє ефективно зберігати облікові записи користувачів, повідомлення, файли, учасників чатів та коди відновлення пароля. Взаємодія з базою даних з серверного C++ коду здійснюється через бібліотеку libpqxx, яка надає типобезпечний інтерфейс для виконання параметризованих SQL-запитів і управління транзакціями. Для уникнення затримок при встановленні нових з'єднань із базою даних реалізований пул з'єднань, що підтримує десять готових до використання з'єднань.

Для розбору та формування JSON-повідомлень, якими обмінюються клієнт і сервер, використовується бібліотека nlohmann/json. Вона надає зручний header-only інтерфейс без зовнішніх залежностей, що спрощує інтеграцію у проєкт і не потребує окремого кроку збірки.

Хешування паролів користувачів виконується алгоритмом Argon2id — сучасним рекомендованим стандартом для захисту паролів, що стійкий до атак за допомогою GPU та ASIC. Це гарантує безпеку облікових даних навіть у разі компрометації бази даних.

Розробка клієнтської частини ведеться у середовищі Qt Creator, яке забезпечує вбудований візуальний редактор форм, інтегрований відладчик, систему автодоповнення коду та зручну інтеграцію з системою збірки CMake. Серверна частина розробляється та компілюється безпосередньо на Linux-сервері з використанням компілятора GCC та системи збірки CMake, що забезпечує гнучке управління залежностями та конфігурацією збірки. Для підготовки дистрибутиву клієнтської частини використовується утиліта windeployqt для автоматичного збору необхідних Qt-бібліотек та інструмент Inno Setup для пакування у Windows-інсталятор.

Використання цього комплексу інструментів і технологій забезпечує створення повнофункціональної, захищеної та масштабованої системи обміну повідомленнями. Комбінація Qt 6, C++, OpenSSL, epoll, PostgreSQL, libpqxx, nlohmann/json та Argon2id дозволяє інтегрувати всі рівні додатку — від графічного інтерфейсу до бази даних — у єдину систему, що забезпечує швидку, безпечну та надійну роботу чату в режимі реального часу.

Висновки до розділу 3

У третьому розділі описано розробку програмного забезпечення системи обміну повідомленнями. Визначено основні абстракції предметної області — «Користувач», «Повідомлення», «Чат», «Сервер», «Сесія» та «База даних», кожна з яких формалізує певний аспект функціонування системи та є основою для подальшого проєктування.

Побудовано UML-діаграми класів та кооперацій, що відображають статичну структуру компонентів і динаміку їх взаємодії під час виконання ключових сценаріїв. Обрано трирівневу клієнт-серверну архітектуру з чітким розподілом відповідальності між клієнтським застосунком, сервером застосунку та сервером бази даних. Розроблено діаграму пакетів, що містить дві підсистеми: клієнтську (ClientGUI, ClientNetwork, ClientModels) та

серверну (ServerNetwork, ServerBusinessLogic, RequestDB, ChatManager, Utils).

Обґрунтовано вибір технологічного стеку: Qt 6 і C++ для клієнтської частини, асинхронний механізм epoll для масштабованої обробки мережевих з'єднань на сервері, OpenSSL для TLS-шифрування каналу передачі даних, PostgreSQL та libpqxx для збереження й обробки інформації, nlohmann/json для серіалізації повідомлень протоколу обміну та алгоритм Argon2id для криптографічно стійкого зберігання паролів.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення коректної та стабільної роботи системи обміну повідомленнями в реальному часі визначено апаратні та програмні вимоги, що враховують особливості архітектури застосунку: асинхронний C++ сервер під управлінням Linux та графічний клієнт на базі Qt 6 під управлінням Windows. Вимоги сформовані окремо для серверної та клієнтської складових, оскільки вони виконуються на різних платформах і мають відмінне навантаження.

Щодо апаратних вимог до серверної частини, вона повинна мати достатню продуктивність для одночасного обслуговування множини TLS-з'єднань, виконання бізнес-логіки та взаємодії з базою даних. Рекомендується використовувати процесор архітектури x86-64 з тактовою частотою не нижче 1,5 ГГц; оскільки серверна частина реалізована як однопотоковий цикл на основі epoll, пріоритет надається частоті, а не кількості ядер, хоча наявність двох і більше ядер бажана для паралельної роботи процесу PostgreSQL. Оперативна пам'ять повинна становити не менше 2 ГБ: це забезпечує стабільну роботу сервера застосунку, пулу з десяти з'єднань із базою даних та самого сервера PostgreSQL. Для зберігання бази даних, завантажених файлів і сертифікатів рекомендується SSD-накопичувач обсягом не менше 20 ГБ, що забезпечить швидкий доступ до даних при обробці запитів. Сервер повинен мати стабільне підключення до мережі Інтернет зі статичним IP-адресою або доменним іменем, оскільки клієнтські застосунки підключаються безпосередньо до нього, а також відкритий порт 5000 для прийому TLS-з'єднань та порт 465 для відправлення електронних листів із кодами скидання пароля через SMTP.

Апаратні вимоги до клієнтської частини є значно скромнішими. Для запуску клієнтського застосунку достатньо сучасного персонального

комп'ютера або ноутбука під управлінням 64-розрядної версії Windows з процесором x86-64 будь-якого виробника та тактовою частотою від 1,0 ГГц. Оперативна пам'ять повинна становити не менше 512 МБ, хоча на практиці рекомендується 2 ГБ і більше для комфортної роботи операційної системи разом із застосунком. Для коректного відображення інтерфейсу чату необхідна роздільна здатність екрана не менше 1280×720 пікселів. Підключення до мережі Інтернет або локальної мережі є обов'язковим для взаємодії із сервером.

Програмні вимоги до серверного середовища включають операційну систему Linux з ядром версії 3.9 або новіше, що підтримує системний виклик epoll. Рекомендованим дистрибутивом є Ubuntu Server 20.04 LTS або новіший. На сервері має бути встановлено: компілятор GCC версії 10 або новіше з підтримкою стандарту C++17; систему збірки CMake версії 3.16 або новіше; сервер баз даних PostgreSQL версії 13 або новіше разом із заголовними файлами для розробки (libpq-dev); бібліотеку libpqxx версії 7 або новіше для взаємодії C++ коду з PostgreSQL; бібліотеку OpenSSL версії 3.x для реалізації TLS-захисту з'єднань та SMTPS-клієнта; бібліотеку libargon2 для хешування паролів за алгоритмом Argon2id; а також бібліотеку nlohmann/json для розбору та формування JSON-повідомлень. Для роботи TLS-захисту необхідна наявність дійсного сертифіката (cert.pem) та закритого ключа (key.pem), що розміщуються у робочій директорії серверного процесу. Для надсилання листів із кодами відновлення пароля потрібен обліковий запис Gmail із увімкненою двофакторною автентифікацією та згенерованим паролем програми.

Програмні вимоги до клієнтської частини є мінімальними для кінцевого користувача, оскільки всі необхідні бібліотеки постачаються разом із застосунком у складі інсталятора. Операційна система — Windows 10 або Windows 11 (64-розрядна версія). Окремого встановлення Qt, OpenSSL або будь-яких інших бібліотек не потрібно: інсталятор, підготовлений за допомогою утиліти windeployqt та інструменту Inno Setup, автоматично

розміщує всі необхідні компоненти у директорії застосунку, зокрема Qt-бібліотеки, плагіни платформи, бібліотеки OpenSSL (libssl-3-x64.dll, libcrypto-3-x64.dll) та середовище виконання MinGW. Для розробки та компіляції клієнтської частини необхідне середовище Qt Creator з встановленим набором Qt 6.5 або новіше для компілятора MinGW 64-bit, а також CMake версії 3.16 або новіше.

Узгодження апаратних та програмних вимог дозволяє забезпечити стабільну та безперебійну роботу системи, мінімізувати затримки при обробці повідомлень та гарантувати надійний захист переданих даних. Вибір сучасного інструментарію, зокрема асинхронного сервера на базі epoll, TLS-шифрування через OpenSSL та реляційної бази даних PostgreSQL, є ключовим для досягнення високого рівня продуктивності та масштабованості системи, а також для забезпечення її подальшого розвитку та збільшення кількості одночасних користувачів.

4.2 Тестування системи

Запустивши систему, потрапляємо на початкову форму “Привітання”. З цієї форми ми можемо перейти на форму реєстрації та форму входу в систему(рис. 4.1).

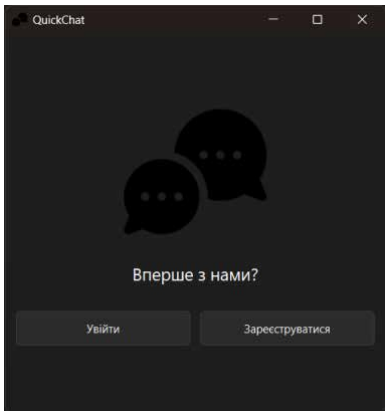

Рис. 4.1 Форма “Привітання”

Зареєструємо нового користувача (Рис. 4.2).

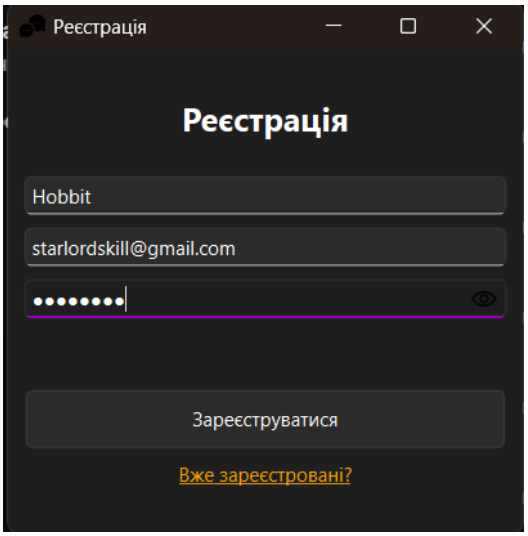

Рис. 4.2 Реєстрація нового користувача

Входимо в систему(Рис. 4.3).

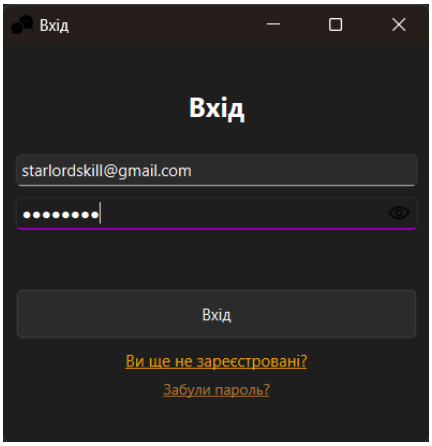

Рис. 4.3 Вхід в систему

Додаємо нового користувача в список чатів (Рис. 4.4).

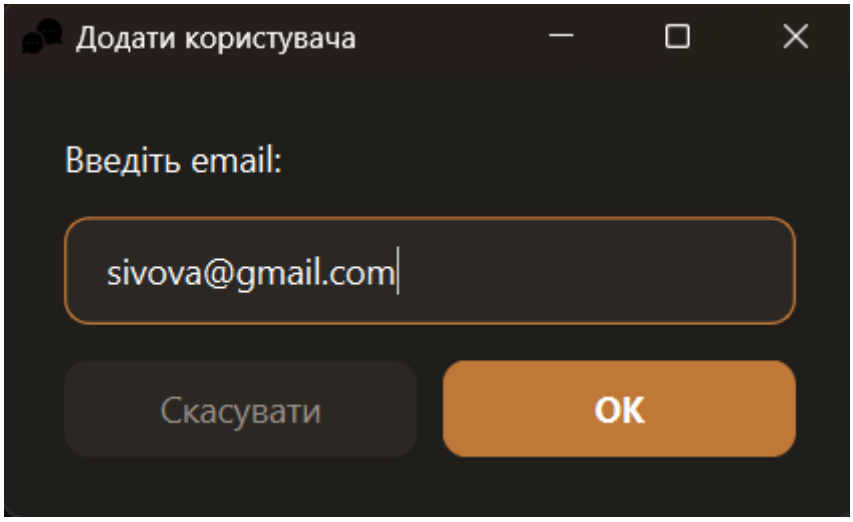

Рис. 4.4 Створення нового чату

Обираємо чат та обмінюємось повідомленнями із іншим учасником чату також можемо видалити або змінити текст повідомлення, а також завантажити та вивантажити файл(Рис. 4.5).

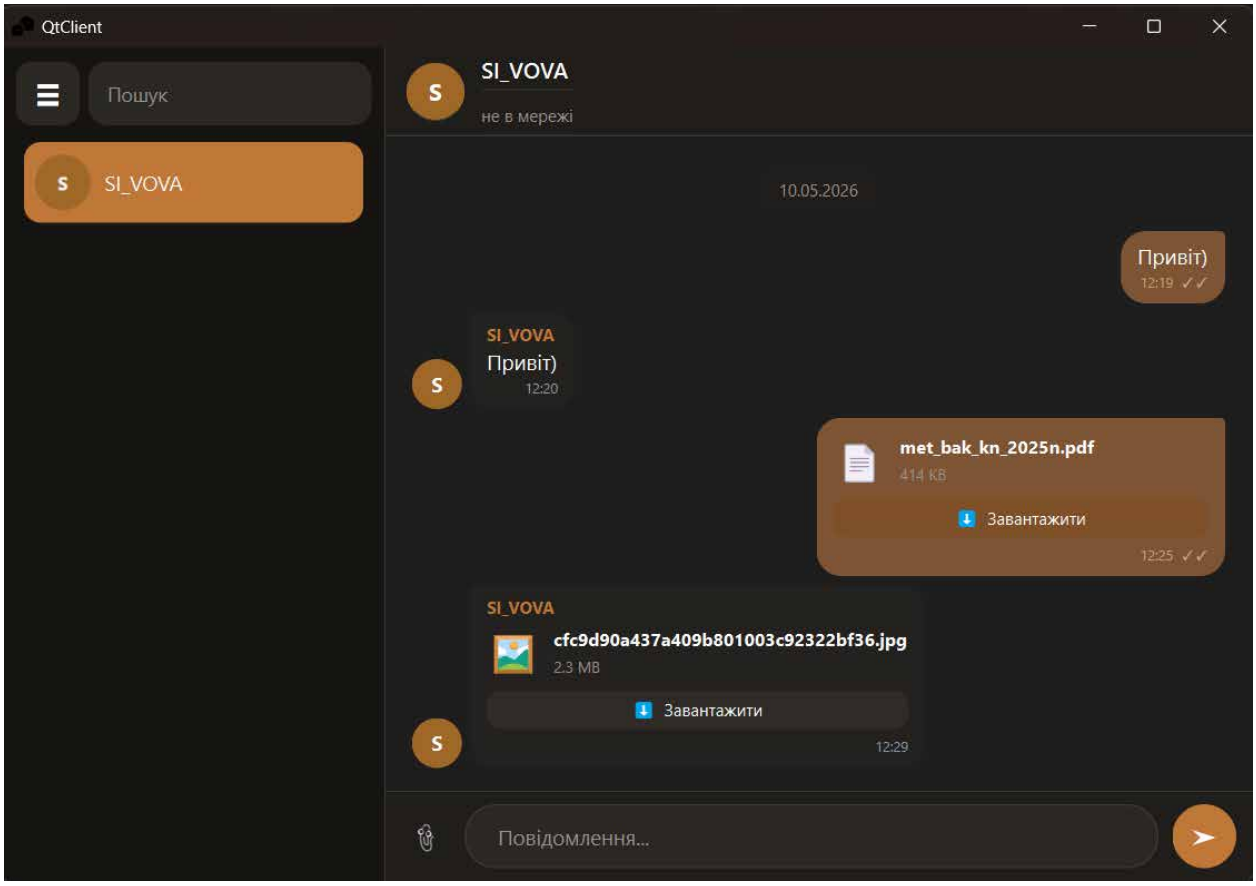

Рис. 4.5 Головна форма програми

Висновки до розділу 4

У четвертому розділі описано впровадження та тестування розробленої системи обміну повідомленнями. Визначено апаратні та програмні вимоги для серверної і клієнтської частин окремо: серверна частина потребує Linux-системи з ядром версії 3.9 або новіше, компілятора GCC 10+, СУБД PostgreSQL 13+, бібліотек OpenSSL 3.x, libpqxx 7+ та libargon2; клієнтська частина розрахована на 64-розрядну Windows 10 або Windows 11 з роздільною здатністю екрана не менше 1280×720 пікселів та підключенням до мережі.

Проведено практичне тестування всіх функціональних модулів системи: продемонстровано роботу форми привітання, реєстрації нового користувача та входу в систему, створення нового чату за електронною адресою, обміну текстовими повідомленнями між учасниками, редагування та видалення повідомлень, а також надсилання і завантаження файлів. Результати тестування підтверджують відповідність реалізованого програмного забезпечення функціональним вимогам, визначеним у першому розділі.

ВИСНОВКИ

У цій бакалаврській роботі було здійснено розробку програмного забезпечення системи обміну повідомленнями в реальному часі з підтримкою захищеного з'єднання, групових чатів та передачі файлів. У вступі обґрунтовано актуальність дослідження, визначено об'єкт та предмет дослідження, а також поставлено мету — створення повнофункціонального чат-застосунку, що забезпечує захищену взаємодію між користувачами в режимі реального часу із підтримкою сучасних комунікаційних функцій.

Було проведено детальний аналіз предметної області, визначено основні терміни, атрибути та взаємозв'язки між ними, а також охарактеризовано існуючі рішення на ринку месенджерів, зокрема Telegram, Discord, що дозволило виділити сильні та слабкі сторони сучасних систем і сформувати вимоги до власного рішення. На основі цього було сформульовано функціональні та нефункціональні вимоги, описані вимоги до інтерфейсу користувача, що забезпечують зручність взаємодії, персоналізацію профілю та безпечне відновлення доступу до облікового запису.

Для моделювання системи використовувалась UML-методика, зокрема діаграми прецедентів, компонентів, діяльності, класів та пакетів. Було розроблено сценарії використання основних варіантів, таких як «Авторизація користувача», «Надсилання повідомлення», «Створення групового чату» та «Відновлення паролю», що дозволило відтворити логіку взаємодії користувачів із системою. Використання діаграм класів та пакетів забезпечило чітку організацію компонентів системи та їх взаємозв'язків, а аналіз абстракцій предметної області дозволив формалізувати основні сутності та зв'язки між ними.

Було здійснено вибір та обґрунтовано архітектуру програмного забезпечення. Застосовано дворівневу клієнт-серверну архітектуру з чітким

розподілом на підсистеми: клієнтська частина відповідає за графічний інтерфейс та мережеву взаємодію, серверна — за маршрутизацію повідомлень, бізнес-логіку та роботу з базою даних. Для реалізації програмного забезпечення обрано інструменти та технології: C++, Qt 6, OpenSSL, epoll, PostgreSQL, libpqxx, nlohmann/json та алгоритм хешування Argon2id. Таке поєднання забезпечило асинхронну обробку з'єднань на сервері, захищену TLS-передачу даних між клієнтом і сервером, надійне зберігання інформації та безпеку облікових даних користувачів.

У рамках роботи було спроєктовано та реалізовано базу даних, що включає таблиці користувачів, чатів, учасників чатів, повідомлень та файлів, а також забезпечено підтримку структурованих запитів і транзакцій. Було розроблено логіку обробки повідомлень користувачів, реалізовано функціонал приватних та групових чатів, передачу та завантаження файлів, редагування та видалення повідомлень, управління профілем і аватаром, а також систему відновлення паролю з надсиланням коду підтвердження на електронну пошту через протокол SMTPS.

Результатом роботи є повнофункціональний захищений месенджер, що складається з клієнтського застосунку для Windows на базі Qt 6 та асинхронного серверного процесу для Linux, які взаємодіють через зашифроване TLS-з'єднання. Застосунок дозволяє користувачам реєструватися, авторизуватися, обмінюватися текстовими повідомленнями та файлами у приватних і групових чатах, відстежувати статус присутності співрозмовників в режимі реального часу та безпечно відновлювати доступ до облікового запису.

Таким чином, виконана робота підтверджує доцільність використання асинхронних мережевих технологій та сучасних криптографічних засобів для побудови захищених комунікаційних систем, демонструє ефективність застосування фреймворку Qt 6 для розробки крос-платформенних графічних застосунків, а отримані результати можуть стати основою для подальшого розвитку системи, зокрема реалізації мобільного

клієнта, наскрізного шифрування повідомлень та масштабування серверної частини для обслуговування більшої кількості користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stevens, W.R., Fenner, B., Rudoff, A.M. UNIX Network Programming, Volume 1: The Sockets Networking API. 3rd Edition. — Addison-Wesley, 2003. — 1024 с.

2. Sommerville, I. Software Engineering. 10th Edition. — Pearson, 2015. — 816 с.

3. Tanenbaum, A.S., Wetherall, D.J. Computer Networks. 5th Edition. — Prentice Hall, 2010. — 960 с.

4. Biryukov, A., Dinu, D., Großschädl, J., Pasalic, E. — Argon2: Memory-Hard Function for Password Hashing – [Електронний ресурс] – Режим доступу: https://github.com/P-H-C/phc-winner-argon2

5. IETF — RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 – [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/html/rfc8446

6. Qt Documentation — Qt Network Module – [Електронний ресурс] – Режим доступу: https://doc.qt.io/qt-6/qtnetwork-index.html

7. Google — Base64 Encoding Explained – [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/Base64

8. libpqxx — The Official C++ Client API for PostgreSQL – [Електронний ресурс] – Режим доступу: https://pqxx.org/development/libpqxx/

9. Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language User Guide. 2nd Edition. — Addison-Wesley, 2005. - 496с.

10. Discord Documentation https://docs.discord.com/developers/reference

11. OpenSSL Project — OpenSSL Documentation – [Електронний ресурс] – Режим доступу: https://www.openssl.org/docs/

12. Qt Documentation — QSslSocket Class – [Електронний ресурс] – Режим доступу: https://doc.qt.io/qt-6/qsslsocket.html

13. PostgreSQL Global Development Group — PostgreSQL 15 Documentation – [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/15/

14. Lohmann, N. — JSON for Modern C++ (nlohmann/json) – [Електронний ресурс] – Режим доступу: https://github.com/nlohmann/json

15. Linux Programmer's Manual — epoll(7) – [Електронний ресурс] – Режим доступу: https://man7.org/linux/man-pages/man7/epoll.7.html

16. CMake Project — CMake Documentation – [Електронний ресурс] – Режим доступу: https://cmake.org/documentation/

17. Inno Setup — Inno Setup Documentation – [Електронний ресурс] – Режим доступу: https://jrsoftware.org/ishelp/

18. IETF — RFC 5321: Simple Mail Transfer Protocol – [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/html/rfc5321

19. OWASP Foundation — OWASP Top Ten Security Risks – [Електронний ресурс] – Режим доступу: https://owasp.org/www-project-top-ten/

20. OWASP Foundation — Password Storage Cheat Sheet – [Електронний ресурс] – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

21. Stroustrup, B. The C++ Programming Language. 4th Edition. — Addison-Wesley, 2013. — 1366 с.

22. Meyers, S. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14. — O'Reilly Media, 2014. — 334 с.

23. Blanchette, J., Summerfield, M. C++ GUI Programming with Qt 4. 2nd Edition. — Prentice Hall, 2008. — 752 с.

24. Qt Documentation — Qt 6 Reference Documentation – [Електронний ресурс] – Режим доступу: https://doc.qt.io/qt-6/

25. Fall, K., Stevens, W.R. TCP/IP Illustrated, Volume 1: The Protocols. 2nd Edition. — Addison-Wesley, 2011. — 1032 с.

26. Viega, J., Messier, M., Chandra, P. Network Security with OpenSSL. — O'Reilly Media, 2002. — 372 с.

27. Momjian, B. PostgreSQL: Introduction and Concepts. — Addison-Wesley, 2001. — 460 с.

28. Date, C.J. An Introduction to Database Systems. 8th Edition. — Pearson, 2003. — 1024 с.

29. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. — Addison-Wesley, 1995. — 395 с.

30. Fowler, M. Patterns of Enterprise Application Architecture. — Addison-Wesley, 2002. — 560 с.

31. Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. 3rd Edition. — Prentice Hall, 2004. — 736 с.

32. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. — Wiley, 1996. — 476 с.

33. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F. Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. — Wiley, 2000. — 666 с.

34. Martin, R.C. Clean Code: A Handbook of Agile Software Craftsmanship. — Prentice Hall, 2008. — 431 с.

35. GCC Team — GCC, the GNU Compiler Collection – [Електронний ресурс] – Режим доступу: https://gcc.gnu.org/

ДОДАТОК А

Код компонента відображення повідомлення

```
// messagewidget.h
#ifndef MESSAGEWIDGET_H
#define MESSAGEWIDGET_H

#include <QWidget>
#include <QLabel>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QPixmap>

struct FileInfo {
   QString name;
   QString storedPath;
   QString mime;
   qint64  size = 0;
   bool isEmpty() const { return name.isEmpty(); }
};

class MessageWidget : public QWidget
{
   Q_OBJECT

public:
   explicit MessageWidget(const QString &displayName,
                          const QString &text,
                          const QString &time,
                          bool isMine,
                          const QString &avatarBg,
                          bool showAvatar,
                          int  messageId,
                          bool isEdited   = false,
                          bool isDeleted  = false,
                          const FileInfo &fileInfo    = FileInfo{},
                          const QPixmap  &avatarPixmap = QPixmap{},
                          QWidget *parent = nullptr);

   void setEdited(bool edited);
   void setDeleted();
   void updateText(const QString &newText);

signals:
   void actionRequested(int msgId, const QString &curText,
                        bool isMine, const QPoint &pos);
   void downloadRequested(const QString &storedPath,
                          const QString &fileName);

protected:
   bool eventFilter(QObject *obj, QEvent *event) override;
```

```
private:
    void installFilterRecursive(QWidget *w);

    int      msgId;
    bool     mine;
    bool     isFileMsg;

    QLabel      *messageLabel      = nullptr;
    QLabel      *editedLabel       = nullptr;
    QWidget     *fileContentWidget = nullptr;
};

#endif // MESSAGEWIDGET_H
```

```
// messagewidget.cpp
#include "messagewidget.h"
#include <QMouseEvent>
#include <QPainter>
#include <QPainterPath>

static QString formatSize(qint64 bytes)
{
    if (bytes < 1024)
        return QString("%1 B").arg(bytes);
    if (bytes < 1024 * 1024)
        return QString("%1 KB").arg(bytes / 1024);
    return QString("%1 MB").arg(bytes / 1024.0 / 1024.0, 0, 'f', 1);
}

static QPixmap makeCircularAvatar(const QPixmap &src, int size)
{
    QPixmap scaled = src.scaled(size, size,
                                Qt::KeepAspectRatioByExpanding,
                                Qt::SmoothTransformation);
    if (scaled.width() > size)
        scaled = scaled.copy((scaled.width() - size) / 2, 0, size, size);
    if (scaled.height() > size)
        scaled = scaled.copy(0, (scaled.height() - size) / 2, size, size);

    QPixmap circle(size, size);
    circle.fill(Qt::transparent);
    QPainter p(&circle);
    p.setRenderHint(QPainter::Antialiasing);
    QPainterPath path;
    path.addEllipse(0, 0, size, size);
    p.setClipPath(path);
    p.drawPixmap(0, 0, scaled);
    return circle;
}

void MessageWidget::installFilterRecursive(QWidget *w)
{
```

```
    w->installEventFilter(this);
    for (QObject *child : w->children())
        if (QWidget *cw = qobject_cast<QWidget*>(child))
            installFilterRecursive(cw);
}

MessageWidget::MessageWidget(const QString &displayName,
                             const QString &text,
                             const QString &time,
                             bool isMine,
                             const QString &avatarBg,
                             bool showAvatar,
                             int  messageId,
                             bool isEdited,
                             bool isDeleted,
                             const FileInfo &fileInfo,
                             const QPixmap  &avatarPixmap,
                             QWidget *parent)
    : QWidget(parent)
    , msgId(messageId)
    , mine(isMine)
    , isFileMsg(!fileInfo.isEmpty())
{
    QHBoxLayout *mainLayout = new QHBoxLayout(this);
    mainLayout->setContentsMargins(4, 2, 4, 2);
    mainLayout->setSpacing(0);

    // Бульбашка повідомлення (батько — this, щоб уникнути
    // появи як окремого вікна до вбудування в макет)
    QWidget *bubble = new QWidget(this);
    bubble->setObjectName("bubble");
    bubble->setMaximumWidth(420);
    bubble->setAttribute(Qt::WA_StyledBackground, true);

    QVBoxLayout *bl = new QVBoxLayout(bubble);
    bl->setContentsMargins(12, 8, 12, 8);
    bl->setSpacing(2);

    // Ім'я відправника (лише для вхідних, перше у групі)
    if (!isMine && showAvatar)
    {
        QLabel *senderLabel = new QLabel(displayName, bubble);
        senderLabel->setStyleSheet(
            "color:#c07838; font-size:12px;"
            "font-weight:bold; background:transparent;");
        bl->addWidget(senderLabel);
    }

    // Вміст: файл або текст
    if (isFileMsg && !isDeleted)
    {
        fileContentWidget = new QWidget(bubble);
```

```
fileContentWidget->setStyleSheet("background:transparent;");
QVBoxLayout *fcl = new QVBoxLayout(fileContentWidget);
fcl->setContentsMargins(0, 4, 0, 4);
fcl->setSpacing(8);

QWidget *infoRow = new QWidget(fileContentWidget);
infoRow->setStyleSheet("background:transparent;");
QHBoxLayout *infoLayout = new QHBoxLayout(infoRow);
infoLayout->setContentsMargins(0, 0, 0, 0);
infoLayout->setSpacing(10);

bool isImage = fileInfo.mime.startsWith("image/");
QLabel *iconLabel = new QLabel(isImage ? "🖼" : "📄", infoRow);
iconLabel->setStyleSheet("font-size:28px; background:transparent;");
iconLabel->setFixedWidth(38);
iconLabel->setAlignment(Qt::AlignVCenter | Qt::AlignHCenter);

QWidget *nameAndSize = new QWidget(infoRow);
nameAndSize->setStyleSheet("background:transparent;");
QVBoxLayout *nsl = new QVBoxLayout(nameAndSize);
nsl->setContentsMargins(0, 0, 0, 0);
nsl->setSpacing(2);

QLabel *nameLabel = new QLabel(fileInfo.name, nameAndSize);
nameLabel->setStyleSheet(
   "color:white; font-size:13px;"
   "font-weight:bold; background:transparent;");
nameLabel->setWordWrap(true);

QLabel *sizeLabel = new QLabel(
   formatSize(fileInfo.size), nameAndSize);
sizeLabel->setStyleSheet(
   "color:#9a9088; font-size:11px; background:transparent;");

nsl->addWidget(nameLabel);
nsl->addWidget(sizeLabel);
infoLayout->addWidget(iconLabel, 0, Qt::AlignVCenter);
infoLayout->addWidget(nameAndSize, 1);
fcl->addWidget(infoRow);

QPushButton *dlBtn =
   new QPushButton("⬇  Завантажити", fileContentWidget);
dlBtn->setCursor(Qt::PointingHandCursor);
dlBtn->setStyleSheet(
   isMine
   ? "QPushButton{background:#7e5028;border:none;"
     "border-radius:10px;color:white;font-size:12px;"
     "padding:6px 14px;} QPushButton:hover{background:#8e6038;}"
   : "QPushButton{background:#302c28;border:none;"
     "border-radius:10px;color:white;font-size:12px;"
     "padding:6px 14px;} QPushButton:hover{background:#403c38;}");
```

```
    QString sPath = fileInfo.storedPath;
    QString fName = fileInfo.name;
    connect(dlBtn, &QPushButton::clicked, this, [=]() {
        emit downloadRequested(sPath, fName);
    });
    fcl->addWidget(dlBtn);
    bl->addWidget(fileContentWidget);

    messageLabel = new QLabel(bubble);
    messageLabel->setVisible(false);
    messageLabel->setWordWrap(true);
    bl->addWidget(messageLabel);
}
else
{
    messageLabel = new QLabel(
        isDeleted ? "Повідомлення видалено" : text, bubble);
    messageLabel->setWordWrap(true);
    messageLabel->setStyleSheet(
        isDeleted
        ? "color:#8899aa; font-size:15px;"
          "font-style:italic; background:transparent;"
        : "color:white; font-size:15px; background:transparent;");
    bl->addWidget(messageLabel);

    editedLabel = new QLabel("(edited)", bubble);
    editedLabel->setStyleSheet(
        isMine
        ? "color:#c0a870; font-size:10px; background:transparent;"
        : "color:#8899aa; font-size:10px; background:transparent;");
    editedLabel->setVisible(isEdited);
    bl->addWidget(editedLabel);
}

QLabel *timeLabel =
    new QLabel(isMine ? (time + "  ✓✓") : time, bubble);
timeLabel->setAlignment(Qt::AlignRight);
timeLabel->setStyleSheet(
    isMine
    ? "color:#c0a870; font-size:10px; background:transparent;"
    : "color:#8899aa; font-size:10px; background:transparent;");
bl->addWidget(timeLabel);

bubble->setStyleSheet(isMine
    ? "#bubble { background-color:#7e5535;"
      " border-top-left-radius:18px; border-top-right-radius:4px;"
      " border-bottom-right-radius:18px;"
      " border-bottom-left-radius:18px; }"
    : "#bubble { background-color:#242220;"
      " border-top-left-radius:4px; border-top-right-radius:18px;"
      " border-bottom-right-radius:18px;"
      " border-bottom-left-radius:18px; }");
```

```
    installFilterRecursive(bubble);

    if (isMine)
    {
        mainLayout->addStretch();
        mainLayout->addWidget(bubble);
        mainLayout->addSpacing(4);
    }
    else
    {
        mainLayout->addSpacing(4);
        if (showAvatar)
        {
            QLabel *avatarLabel = new QLabel(this);
            avatarLabel->setFixedSize(36, 36);
            avatarLabel->setAlignment(Qt::AlignCenter);
            if (!avatarPixmap.isNull())
            {
                avatarLabel->setPixmap(
                    makeCircularAvatar(avatarPixmap, 36));
                avatarLabel->setStyleSheet("background:transparent;");
            }
            else
            {
                QString initial = displayName.isEmpty()
                    ? "?" : QString(displayName[0]).toUpper();
                avatarLabel->setText(initial);
                avatarLabel->setStyleSheet(QString(
                    "QLabel { background-color:%1; border-radius:18px;"
                    " color:white; font-size:14px;"
                    " font-weight:bold; }").arg(avatarBg));
            }
            mainLayout->addWidget(avatarLabel, 0, Qt::AlignBottom);
        }
        else { mainLayout->addSpacing(36); }

        mainLayout->addSpacing(6);
        mainLayout->addWidget(bubble);
        mainLayout->addStretch();
    }
}

bool MessageWidget::eventFilter(QObject *obj, QEvent *event)
{
    Q_UNUSED(obj)
    if (event->type() == QEvent::MouseButtonPress)
    {
        auto *me = static_cast<QMouseEvent*>(event);
        if (me->button() == Qt::RightButton)
        {
            QString curText = messageLabel
```

```
            ? messageLabel->text() : QString();
        emit actionRequested(msgId, curText, mine, QCursor::pos());
      }
   }
   return false;
}

void MessageWidget::setEdited(bool edited)
{
   if (!editedLabel || isFileMsg) return;
   if (edited) { editedLabel->setText("(edited)"); editedLabel->show(); }
}

void MessageWidget::setDeleted()
{
   if (fileContentWidget) fileContentWidget->hide();
   if (messageLabel)
   {
      messageLabel->setText("Повідомлення видалено");
      messageLabel->setStyleSheet(
         "color:#8899aa; font-size:15px;"
         "font-style:italic; background:transparent;");
      messageLabel->show();
   }
}

void MessageWidget::updateText(const QString &newText)
{
   if (isFileMsg) return;
   messageLabel->setText(newText);
   messageLabel->setStyleSheet(
      "color:white; font-size:15px; background:transparent;");
   setEdited(true);
}
```

ДОДАТОК Б

Код форми авторизації клієнтської частини

```
// mainwindow.h
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLineEdit>
#include <QPushButton>
#include <QLabel>
#include <QSslSocket>

class MainWindow : public QMainWindow
{
   Q_OBJECT

public:
   explicit MainWindow(QWidget *parent = nullptr);
   ~MainWindow();

private slots:
   void connectToServer();
   void connected();
   void readData();
   void socketError(QAbstractSocket::SocketError);
   void sslErrors(const QList<QSslError>&);

private:
   void resetStyles();

   QLineEdit   *emailEdit;
   QLineEdit   *passEdit;
   QPushButton *loginButton;
   QPushButton *registerLink;
   QPushButton *forgotPasswordLink;
   QLabel      *errorLabel;
   QSslSocket  *socket;
};

#endif // MAINWINDOW_H
// mainwindow.cpp
#include "mainwindow.h"
#include "chatwindow.h"
#include "registerwindow.h"
#include "forgotpassworddialog.h"
#include "config.h"

#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
```

```
#include <QAction>
#include <QIcon>
#include <QJsonDocument>
#include <QJsonObject>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
{
    QWidget *central = new QWidget(this);
    setCentralWidget(central);

    QVBoxLayout *mainLayout = new QVBoxLayout(central);

    QLabel *title = new QLabel("Вхід", this);
    title->setAlignment(Qt::AlignCenter);
    title->setStyleSheet("font-size: 20px; font-weight: bold;");

    emailEdit = new QLineEdit(this);
    emailEdit->setPlaceholderText("Email");

    passEdit = new QLineEdit(this);
    passEdit->setPlaceholderText("Пароль");
    passEdit->setEchoMode(QLineEdit::Password);

    // Кнопка показу/приховання пароля
    QAction *togglePassword = new QAction(this);
    togglePassword->setIcon(QIcon(":/icons/icons/eye.png"));
    passEdit->addAction(togglePassword, QLineEdit::TrailingPosition);
    connect(togglePassword, &QAction::triggered, this, [=]()
    {
        if (passEdit->echoMode() == QLineEdit::Password)
        {
            passEdit->setEchoMode(QLineEdit::Normal);
            togglePassword->setIcon(
                QIcon(":/icons/icons/eye_closed.png"));
        }
        else
        {
            passEdit->setEchoMode(QLineEdit::Password);
            togglePassword->setIcon(QIcon(":/icons/icons/eye.png"));
        }
    });

    errorLabel = new QLabel(this);
    errorLabel->setStyleSheet("color: orange; font-size:11px;");
    errorLabel->setAlignment(Qt::AlignCenter);

    loginButton = new QPushButton("Вхід", this);
    loginButton->setFixedHeight(40);

    registerLink = new QPushButton("Ви ще не зареєстровані?", this);
    registerLink->setFlat(true);
```

```
registerLink->setCursor(Qt::PointingHandCursor);
registerLink->setStyleSheet(
    "QPushButton { color:orange; text-decoration:underline;"
    " border:none; background:transparent; }"
    "QPushButton:hover { color:#ffb347; }");

forgotPasswordLink = new QPushButton("Забули пароль?", this);
forgotPasswordLink->setFlat(true);
forgotPasswordLink->setCursor(Qt::PointingHandCursor);
forgotPasswordLink->setStyleSheet(
    "QPushButton { color:#c07830; text-decoration:underline;"
    " border:none; background:transparent; font-size:11px; }"
    "QPushButton:hover { color:#e07820; }");

QHBoxLayout *linkLayout = new QHBoxLayout();
linkLayout->addStretch();
linkLayout->addWidget(registerLink);
linkLayout->addStretch();

QHBoxLayout *forgotLayout = new QHBoxLayout();
forgotLayout->addStretch();
forgotLayout->addWidget(forgotPasswordLink);
forgotLayout->addStretch();

mainLayout->addStretch();
mainLayout->addWidget(title);
mainLayout->addSpacing(15);
mainLayout->addWidget(emailEdit);
mainLayout->addWidget(passEdit);
mainLayout->addSpacing(5);
mainLayout->addWidget(errorLabel);
mainLayout->addSpacing(10);
mainLayout->addWidget(loginButton);
mainLayout->addLayout(linkLayout);
mainLayout->addLayout(forgotLayout);
mainLayout->addStretch();

// TLS-сокет
socket = new QSslSocket();
connect(socket, &QSslSocket::connected,
        this, &MainWindow::connected);
connect(socket, &QSslSocket::readyRead,
        this, &MainWindow::readData);
connect(socket, &QSslSocket::errorOccurred,
        this, &MainWindow::socketError);
connect(socket, &QSslSocket::sslErrors,
        this, &MainWindow::sslErrors);

connect(loginButton, &QPushButton::clicked,
        this, &MainWindow::connectToServer);

connect(registerLink, &QPushButton::clicked, this, [=]()
```

```
    {
        RegisterWindow *regWin = new RegisterWindow();
        regWin->setAttribute(Qt::WA_DeleteOnClose);
        regWin->show();
        this->close();
    });

    connect(forgotPasswordLink, &QPushButton::clicked, this, [=]()
    {
        ForgotPasswordDialog *dlg = new ForgotPasswordDialog(this);
        dlg->setAttribute(Qt::WA_DeleteOnClose);
        dlg->open();
    });

    setWindowTitle("Вхід");
    setFixedSize(320, 295);
    setWindowIcon(QIcon(":/icons/icons/icon_chat.png"));
}

MainWindow::~MainWindow() {}

void MainWindow::resetStyles()
{
    QString normal = "border:none; border-bottom:1px solid gray;";
    emailEdit->setStyleSheet(normal);
    passEdit->setStyleSheet(normal);
    errorLabel->clear();
}

void MainWindow::connectToServer()
{
    resetStyles();
    QString email    = emailEdit->text();
    QString password = passEdit->text();

    if (email.isEmpty() || password.isEmpty())
    {
        emailEdit->setStyleSheet(
            "border:none; border-bottom:2px solid orange;");
        passEdit->setStyleSheet(
            "border:none; border-bottom:2px solid orange;");
        errorLabel->setText("Введіть email та пароль");
        return;
    }

    if (socket->state() != QAbstractSocket::UnconnectedState)
        socket->abort();

    loginButton->setEnabled(false);
    socket->connectToHostEncrypted(SERVER_IP, SERVER_PORT);
}
```

```
void MainWindow::connected()
{
   QJsonObject obj;
   obj["type"]     = "login";
   obj["email"]    = emailEdit->text();
   obj["password"] = passEdit->text();
   socket->write(
      QJsonDocument(obj).toJson(QJsonDocument::Compact) + "\n");
}

void MainWindow::readData()
{
   static QByteArray buffer;
   buffer.append(socket->readAll());

   while (true)
   {
      int index = buffer.indexOf('\n');
      if (index == -1) break;

      QByteArray msg = buffer.left(index);
      buffer.remove(0, index + 1);

      QJsonDocument doc = QJsonDocument::fromJson(msg);
      if (!doc.isObject()) continue;

      QJsonObject obj  = doc.object();
      QString     type = obj["type"].toString();

      if (type == "login_result")
      {
         if (obj["status"].toString() == "ok")
         {
            disconnect(socket, &QSslSocket::readyRead,
                       this, &MainWindow::readData);

            ChatWindow *chat =
               new ChatWindow(socket, emailEdit->text());
            chat->show();
            this->hide();
         }
         else
         {
            errorLabel->setText("Невірний email або пароль");
            loginButton->setEnabled(true);
         }
      }
   }
}

void MainWindow::socketError(QAbstractSocket::SocketError)
{
```

```
    errorLabel->setText("Сервер недоступний");
    loginButton->setEnabled(true);
}

void MainWindow::sslErrors(const QList<QSslError>&)
{
    socket->ignoreSslErrors(); // self-signed сертифікат
}
```

ДОДАТОК В

SQL-скрипт створення бази даних

```
-- ============================================================
-- Скрипт створення бази даних системи обміну повідомленнями
-- СУБД: PostgreSQL 13+
-- ============================================================

-- Створення бази даних та користувача
-- (виконується від імені суперкористувача postgres)
-- CREATE USER chat_user WITH PASSWORD 'your_password';
-- CREATE DATABASE chat_db OWNER chat_user;
-- GRANT ALL PRIVILEGES ON DATABASE chat_db TO chat_user;

-- ============================================================
-- Таблиця користувачів
-- ============================================================
CREATE TABLE IF NOT EXISTS users (
    id            SERIAL        PRIMARY KEY,
    email         VARCHAR(255)  NOT NULL UNIQUE,
    username      VARCHAR(100)  NOT NULL,
    password_hash TEXT          NOT NULL,
    avatar_path   TEXT,
    reset_code    VARCHAR(6),
    reset_code_exp TIMESTAMP,
    last_active   TIMESTAMP     DEFAULT NOW(),
    created_at    TIMESTAMP     DEFAULT NOW()
);

-- ============================================================
-- Таблиця чатів
-- ============================================================
CREATE TABLE IF NOT EXISTS chats (
    id         SERIAL      PRIMARY KEY,
    type       VARCHAR(10) NOT NULL DEFAULT 'private'
               CHECK (type IN ('private', 'group')),
    name       VARCHAR(255),          -- назва лише для групових
    creator_id INTEGER     REFERENCES users(id) ON DELETE SET NULL,
    created_at TIMESTAMP   DEFAULT NOW()
);

-- ============================================================
-- Таблиця учасників чатів (зв'язок багато-до-багатьох)
-- ============================================================
CREATE TABLE IF NOT EXISTS chat_participants (
    id         SERIAL      PRIMARY KEY,
    chat_id    INTEGER     NOT NULL
               REFERENCES chats(id) ON DELETE CASCADE,
    user_id    INTEGER     NOT NULL
               REFERENCES users(id) ON DELETE CASCADE,
    role       VARCHAR(10) NOT NULL DEFAULT 'member'
               CHECK (role IN ('admin', 'member')),
    joined_at  TIMESTAMP   DEFAULT NOW(),
```

```
    UNIQUE (chat_id, user_id)
);

-- ============================================================
-- Таблиця повідомлень
-- ============================================================
CREATE TABLE IF NOT EXISTS messages (
    id          SERIAL      PRIMARY KEY,
    chat_id     INTEGER     NOT NULL
                REFERENCES chats(id) ON DELETE CASCADE,
    sender_id   INTEGER
                REFERENCES users(id) ON DELETE SET NULL,
    content     TEXT,
    type        VARCHAR(10) NOT NULL DEFAULT 'text'
                CHECK (type IN ('text', 'file')),
    is_edited   BOOLEAN     NOT NULL DEFAULT FALSE,
    is_deleted  BOOLEAN     NOT NULL DEFAULT FALSE,
    created_at  TIMESTAMP   DEFAULT NOW()
);

-- ============================================================
-- Таблиця файлів (метадані вкладень)
-- ============================================================
CREATE TABLE IF NOT EXISTS files (
    id              SERIAL          PRIMARY KEY,
    message_id      INTEGER         NOT NULL
                    REFERENCES messages(id) ON DELETE CASCADE,
    original_name   VARCHAR(255)    NOT NULL,
    stored_path     TEXT            NOT NULL,
    mime_type       VARCHAR(100),
    size_bytes      INTEGER         NOT NULL DEFAULT 0,
    uploaded_at     TIMESTAMP       DEFAULT NOW()
);

-- ============================================================
-- Таблиця статусів прочитання повідомлень
-- ============================================================
CREATE TABLE IF NOT EXISTS message_status (
    id          SERIAL      PRIMARY KEY,
    message_id  INTEGER     NOT NULL
                REFERENCES messages(id) ON DELETE CASCADE,
    user_id     INTEGER     NOT NULL
                REFERENCES users(id) ON DELETE CASCADE,
    is_read     BOOLEAN     NOT NULL DEFAULT FALSE,
    read_at     TIMESTAMP,
    UNIQUE (message_id, user_id)
);

-- ============================================================
-- Індекси для прискорення вибіркових запитів
-- ============================================================
```

```
-- Завантаження історії повідомлень конкретного чату
CREATE INDEX idx_messages_chat_id
   ON messages(chat_id);

-- Пошук повідомлень за відправником
CREATE INDEX idx_messages_sender_id
   ON messages(sender_id);

-- Отримання файлів, прив'язаних до повідомлення
CREATE INDEX idx_files_message_id
   ON files(message_id);

-- Перевірка статусу прочитання за повідомленням
CREATE INDEX idx_status_message_id
   ON message_status(message_id);

-- Перевірка статусу прочитання за користувачем
CREATE INDEX idx_status_user_id
   ON message_status(user_id);

-- ============================================================
-- Кінець скрипту
-- ============================================================
```

ДОДАТОК Д

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, ___Попатенко Юрій Миколайович_____, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення системи спілкування за архітектурою клієнт-сервер» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що:
    1. [ ] інструменти генеративного ШІ не використовувалися.
    2. [✓] інструменти ШІ ChatGPT,Claude були використані для:
        1. [✓] перекладу іншомовних джерел;
        2. [✓] стилістичного редагування та перевірки граматики;
        3. [✓] допомоги у написанні/відлагодженні програмного коду;
        4. [✓] інше: _Пошук та перевірка інформації_____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

« » _____ 2026 р. _____ **Юрій ПОПАТЕНКО**

(підпис)