МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри _____Є. О. Давиденко

підпис

«__»_____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

РОЗРОБКА МЕСЕНДЖЕРА З ФУНКЦІЄЮ ЛОКАЛЬНОГО ЗБЕРЕЖЕННЯ ІСТОРІЇ ЛИСТУВАННЯ

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 409.2121001

Здобувачка _____ М. Ю. Коваленко

підпис

«__»_____ 2024 р.

Керівник канд. пед. наук, доцент _____ К. О. Кірей

підпис

«__»_____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__»_____ 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____Є. О. Давиденко

«___» _____20___ р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано студентці групи 409 факультету комп'ютерних наук

Коваленко Мілані Юріївні

(прізвище, ім'я, по батькові студента)

1. Тема кваліфікаційної роботи

Розробка месенджера з функцією локального збереження історії листування

Затверджена наказом по ЧНУ від «22» грудня 2023 р. № 269

2. Строк представлення кваліфікаційної роботи «___» _____ 20___ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є месенджер з функцією локального збереження історії листування

4. Перелік питань, що підлягають розробці:

- дослідження предметної області та аналіз наявних аналогів;
- формування специфікації вимог до програмного забезпечення;
- моделювання та проєктування програмного забезпечення;
- розробка програмної частини коду сервера;
- розробка клієнтської програмної частини коду.

5. Перелік графічних матеріалів
Презентація

6. Завдання до спеціальної частини
Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алєксєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи канд. пед. наук, доцент Кірей Катерина Олександрівна

(посада, прізвище, ім'я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

Коваленко Мілана Юріївна

(прізвище, ім'я, по батькові студента)

_____

(підпис)

Дата видачі завдання «____» _____ 20___р.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

Тема: Розробка месенджера з функцією локального збереження історії листування

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КРБ	06.12.23	07.12.23	виконано
2.	Огляд літератури за темою роботи	07.02.24	12.02.34	виконано
3.	Складання календарного плану КРБ	08.04.24	09.04.24	виконано
4.	Аналіз предметної області	09.04.24	11.04.24	виконано
5.	Розробка проєктних рішень	12.04.24	20.04.24	виконано
6.	Моделювання та конструювання ПЗ	21.04.24	04.05.24	виконано
7.	Кодування, аналіз результату розробленого ПЗ	05.05.24	27.05.24	виконано
8.	Розробка спеціальної частини з охорони праці	28.05.24	30.05.24	виконано
9.	Оформлення КРБ та презентації	31.05.24	01.05.24	виконано
10.	Відгук керівника КРБ	02.06.24	02.06.24	виконано
11.	Попередній захист	03.06.24	05.06.24	виконано
12.	Завершення оформлення КРБ та презентації	06.06.24	14.06.24	виконано
13.	Рецензування	15.06.24	16.06.24	виконано
14.	Захист кваліфікаційної роботи	25.06.24	25.06.24	виконано

Розробила студентка Коваленко Мілана Юріївна _____

(прізвище, ім'я, по батькові) *(підпис)*

«08» квітня 2024 р.

Керівник роботи

канд. пед. наук, доцент Кірей Катерина Олександрівна _____

(посада, прізвище, ім'я, по батькові) *(підпис)*

«08» квітня 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Розробка месенджера з функцією локального збереження історії листування»

Студентка 409 гр.: Коваленко Мілана Юріївна

Керівник: канд. пед. наук, доцент Кірей К. О.

Кваліфікаційну роботу бакалавра присвячено розробці месенджера з функцією локального збереження історії листування. Актуальність полягає у зростаючій потребі мати засоби обміну інформацією, як діловою, так і особистою, будучи незалежним від місця перебування.

Об’єктом кваліфікаційної роботи є процес комунікації шляхом обміну текстовими повідомленнями.

Предметом кваліфікаційної роботи є організація середовища для обміну тестовими повідомленнями між користувачами з використанням PyQt та криптографічної бібліотеки cryptography.

Метою кваліфікаційної роботи є розробка месенджера з функцією локального збереження історії листування для підвищення безпеки та зручності комунікації між людьми.

Кваліфікаційна робота бакалавра складається з фахової частини і спеціальної частини з охорони праці. Фахова частина в свою чергу поділяється на вступ, чотири розділи, висновки та переліку джерел посилання.

У вступі описується актуальність теми роботи, зазначено об’єкт, предмет, а також мета й завдання, які необхідно виконати для досягнення цієї мети.

У першому розділі проводиться аналіз предметної області. Також оглянуто наявні аналоги, здійснено їх порівняння на основі функціональних можливостей, оцінено переваги та недоліки. У результаті було складено специфікацію вимог до програмного забезпечення, що розробляється, наведено макет графічного інтерфейсу користувача.

У другому розділі описано процес моделювання програмного забезпечення, шляхом опису сценаріїв використання, демонстрації діаграми варіантів використання, діяльності, розгортання.

У третьому розділі описана архітектура системи, що розробляється та надана інформація про технології, які будуть використовуватись для реалізації програмного забезпечення.

У четвертому розділі демонструється процес кодування програмного забезпечення та здійснюється огляд результатів.

У висновках проводиться аналіз виконаної роботи та отриманих результатів.

Кваліфікаційна робота бакалавра викладена на 60 сторінок, містить 4 розділи, 35 ілюстрацій, 8 таблиць, 14 джерел в переліку посилань.

Ключові слова: *месенджер, розробка на Python, застосунок для обміну повідомленнями, локальне збереження історії листування, шифрування передачи даних, PyQt.*

ABSTRACT

of the Bachelor’s Thesis

«Development of a messenger with the function of local storage of chat history»

Supervisor: Ph.D. Sc., Associate Professor Kirei K. O.

The Bachelor’s Thesis is devoted to the development of a messenger with the function of local storage of chat history. The relevance is the increasing need to have a tool for exchanging information, both business and personal, while being independent of location.

The object of the qualification work is the process of communication through text messaging.

The subject of the qualification work is the organization of an environment for exchanging test messages between users using PyQt and the cryptographic library cryptography.

The purpose of the qualification work is to develop a messenger with the function of local storage of the history of correspondence to increase the security and convenience of communication between people.

The Bachelor's Thesis consists of a professional part and a special part on labor protection. The professional part, in turn, is divided into an introduction, four sections, conclusions and a list of references.

The introduction describes the relevance of the topic of the work, indicates the object, subject, as well as the goal and tasks to be performed to achieve this goal.

The first chapter analyzes the subject area. Existing analogs are also reviewed, compared based on functionality, and their advantages and disadvantages are evaluated. As a result, a specification of requirements for the software being developed was drawn up, and a mockup of the graphical user interface was presented.

The second section describes the process of software modeling by describing use cases, demonstrating a diagram of use cases, activities, and deployment.

The third section describes the architecture of the system under development and provides information about the technologies that will be used to implement the software.

The fourth section demonstrates the software coding process and reviews the results.

The conclusions analyze the work performed and the results obtained.

The qualification work is presented on 60 pages, 4 sections, 35 illustrations, 8 tables, 14 sources in the list of references.

Keywords: *messenger, Python programming, messaging application, local storage of chat history, data encryption, PyQt.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
КРБ	–	кваліфікаційна робота бакалавра
ПЗ	–	програмне забезпечення
ПК	–	персональний комп’ютер
СКБД	–	система керування базами даних
ORM	–	object-relational mapping
SQL	–	structured query language
UI	–	user interface
UML	–	unified modeling language

ВСТУП

Актуальність теми полягає у зростаючій потребі в сучасному світі мати засоби обміну інформацією, як діловою, так і особистою, будучи незалежним від місця перебування. Проблема нестачі часу, яка виникає через навантаженість людей власними справами, створює тенденцію прихильності до текстової форми спілкування, яка надає можливість як читати, так і відправляти повідомлення коли завгодно. Відправлені та отримані повідомлення, за потреби, завжди можна зручно перечитати та цим згадати про важливу інформацію, на відміну від комунікації з використанням дзвінків.

Використовувати смс-повідомлення є фінансово невигідним та неактуальним у суспільстві, а ідентифікація користувача для інших людей та можливість спілкування залежить від наявності сім-карти у пристрої. А електронна пошта не надає можливість обміну повідомленнями в реальному часі, що робить її непридатною для використовування як універсального засобу для текстової комунікації.

Потреба у одночасному обміні інформацією з багатьма людьми, ефективній координації, прийнятті спільних рішень, можна вирішити з допомогою використання месенджеру. Проте, тут є проблема забезпечення безпеки спілкування, адже не кожен месенджер має функцію локального збереження історії листування, яка дозволяє зменшити ризик доступу до конфіденційної інформації з боку сторонніх осіб, оскільки дані зберігаються лише на пристрої користувача. Отже розробка месенджера з функцією локального збереження історії листування є актуальним завданням.

Об’єктом кваліфікаційної роботи є процес комунікації шляхом обміну текстовими повідомленнями.

Предметом кваліфікаційної роботи є організація середовища для обміну тестовими повідомленнями між користувачами з використанням PyQt та криптографічної бібліотеки cryptography.

Метою кваліфікаційної роботи є розробка месенджера з функцією локального збереження історії листування для підвищення безпеки та зручності

комунікації між людьми.

Для досягнення поставленої мети необхідно виконати наступні **завдання**:

- дослідження предметної області та аналіз наявних аналогів;
- формування специфікації вимог до програмного забезпечення;
- моделювання та проєктування програмного забезпечення;
- розробка програмної частини коду сервера;
- розробка клієнтської програмної частини коду.

1 АНАЛІЗ МЕСЕНДЖЕРІВ

Перед тим як розпочати розробку програмного забезпечення, необхідно проаналізувати наявні аналоги. Це може допомогти уникнути недоліків уже наявного ПЗ, тим самим розробивши краще рішення, яке потенційно може підвищити шанс на привернення уваги майбутніх користувачів. Також аналізуючи наявні аналоги, можна прийти до висновку який функціонал користується найбільшим попитом, що буде корисним у складанні специфікації вимог.

1.1 Аналіз прикладів аналогічних систем

Нижче наведений результат аналізу такого наявного ПЗ: WhatsApp Messenger (див. табл. 1.1), Signal (див. табл. 1.2), Telegram (див. табл. 1.3).

Таблиця 1.1 – Аналіз системи WhatsApp Messenger

Назва	WhatsApp Messenger [1]
Розробник	Meta
Архітектура	client-server
Основні мови реалізації	Erlang, Java, Swift, C#, Javascript.
Основні функції	1) підтвердження доставки повідомлення; 2) наявність групових чатів; 3) наскрізне шифрування повідомлень; 4) резервне копіювання та відновлення даних; 5) обмін мультимедіа.
Переваги	1) безкоштовність; 2) економне використання інтернет-трафіку та пам'яті пристрою за допомогою стикання мультимедійних даних перед відправкою; 3) наявність end-to-end шифрування повідомлень.
Недоліки	1) не шифруються метадані повідомлень; 2) обмеження на кількість учасників в групі (256); 3) для спілкування необхідно ділитись мобільним номером.
Вебсайт	https://www.whatsapp.com

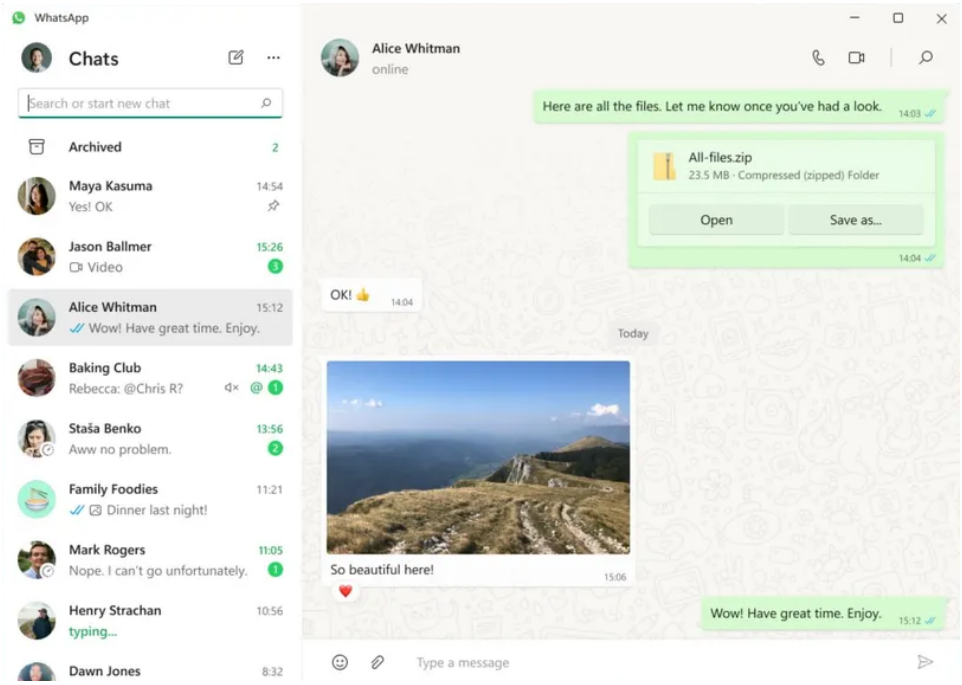

Рисунок 1.1 – Приклад інтерфейсу WhatsApp Messenger

Таблиця 1.2 – Аналіз системи Signal

Назва	Signal [2]
Розробник	Signal Foundation
Архітектура	client-server
Основні мови реалізації	JavaScript, Java, Objective-C, C
Основні функції	1) можливість автоматичного видалення повідомлень; 2) наявність групових чатів; 3) наскрізне шифрування повідомлень; 4) шифровані дзвінки; 5) обмін мультимедіа.
Переваги	1) безкоштовність; 2) високий рівень безпеки; 3) наявність end-to-end шифрування повідомлень; 4) відсутність реклами та збору даних; 5) відкритий код; 6) дані повідомлень зберігаються лише на пристроях користувачів.
Недоліки	1) невелика кількість користувачів, порівняно з іншими месенджерами; 2) необхідний номер телефону для реєстрації.
Вебсайт	https://signal.org

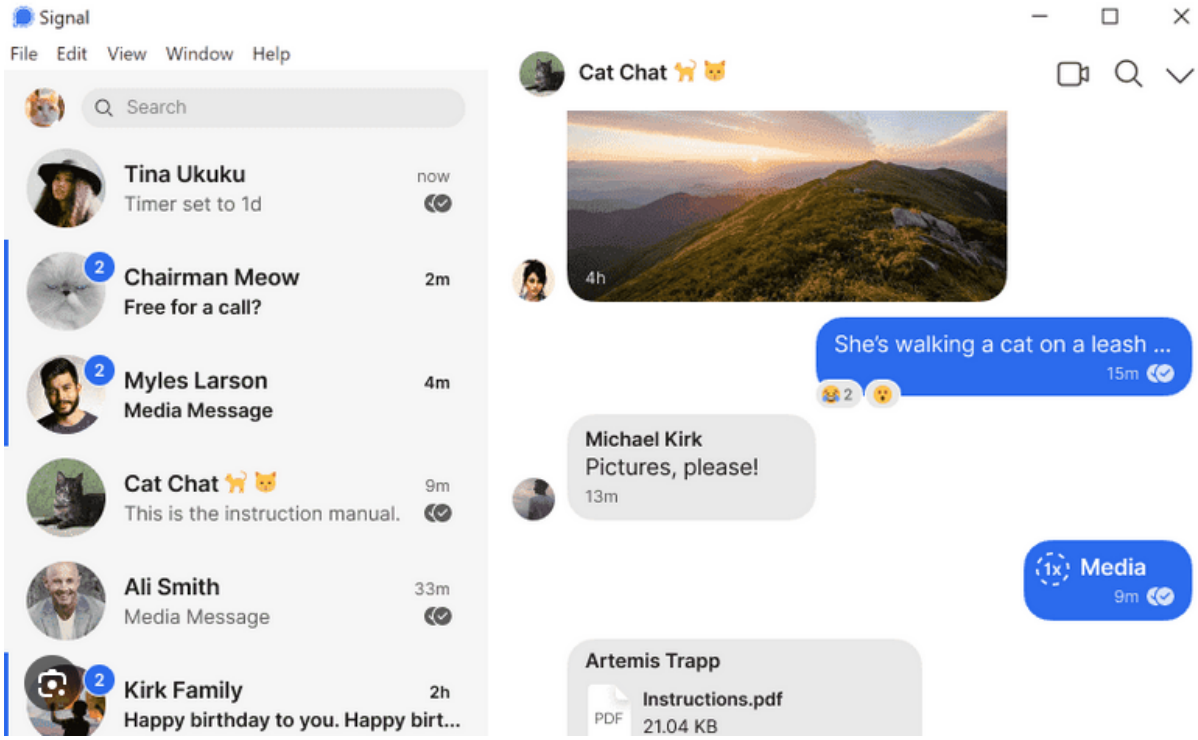

Рисунок 1.2 – Приклад інтерфейсу Signal

Таблиця 1.3 – Аналіз системи Telegram

Назва	Telegram [3]
Розробник	Telegram Messenger LLP
Архітектура	client-server
Основні мови реалізації	C++, Swift, C, Java, Objective-C, NodeJS, AngularJS
Основні функції	1) підтвердження доставки повідомлення; 2) наявність групових чатів; 3) можливість наскрізного шифрування повідомлень; 4) підтримка ботів та інтеграція з різними сервісами; 5) обмін мультимедіа.
Переваги	1) безкоштовність; 2) підтримка ботів та інтеграція з різними сервісами; 3) велика кількість користувачів, порівняно з іншими месенджерами.
Недоліки	1) інформація чатів зберігається на серверах; 2) наскрізне шифрування працює тільки в спеціально створених секретних чатах.
Вебсайт	https://telegram.org

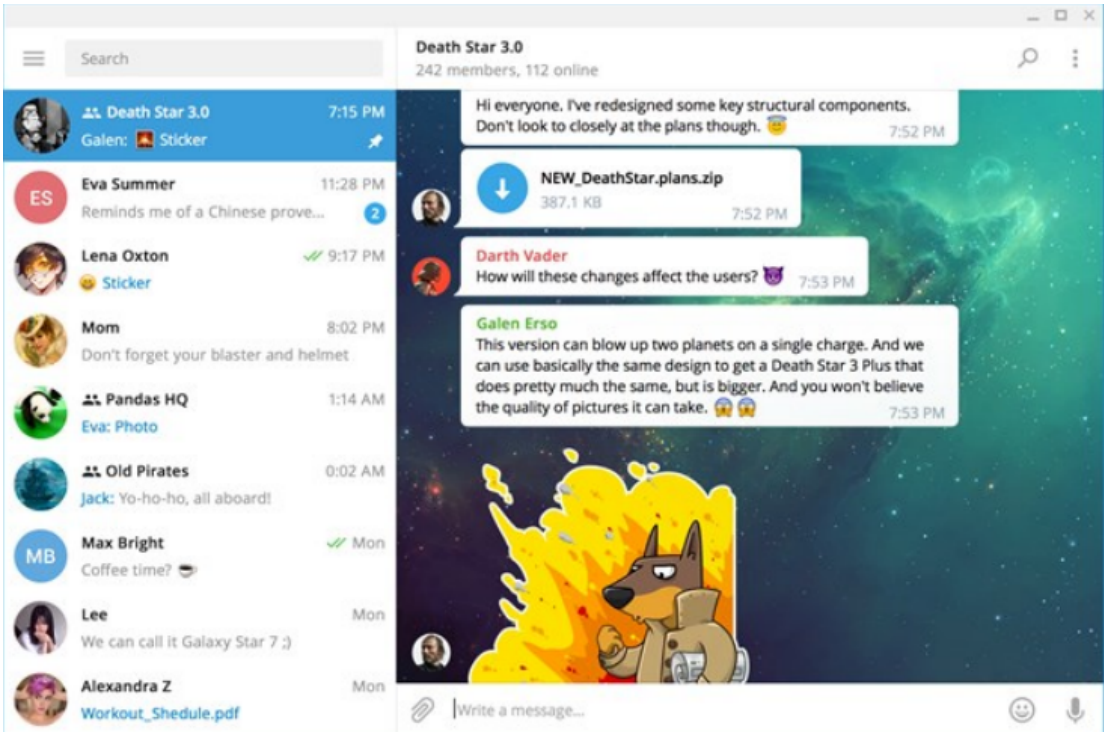

Рисунок 1.3 – Приклад інтерфейсу Telegram

З огляду на результати аналізу, можна зробити такі висновки:

– WhatsApp Messenger є популярним месенджером, але він має обмеження щодо конфіденційності, оскільки використовує стандартний протокол шифрування;

– Signal відомий своєю високою безпекою та шифруванням, але його популярність менша, ніж у WhatsApp;

– Telegram має велику кількість користувачів, але його шифрування не є настільки надійним, як у Signal.

1.2 Специфікації вимог до програмного забезпечення

Основне призначення системи – обмін повідомленнями та медіафайлами, надаючи користувачам можливість швидкого та безпечного спілкування. Окрім того, з огляду на актуальність забезпечення конфіденційності, ПЗ буде інтегрувати функцію локального збереження історії листування, що зменшить ризик несанкціонованого доступу до особистих даних. Таким чином, це стане відповіддю на сучасні вимоги до швидкості, зручності та безпеки в обміні інформацією. Короткий та загальний опис проєкту що розробляється наведено нижче (табл. 1.4)

Таблиця 1.4 – Опис проєкту що розробляється

Основні задачі	1) реєстрація та авторизація; 2) редагування профілю та налаштувань; 3) пошук інших профілів за нікнеймом; 4) керування контактами та списком заблокованих користувачів; 5) обмін текстовими повідомленнями та файлами; 6) завантаження надісланих файлів на пристрій; 7) резервне копіювання та відновлення даних листування; 8) шифрування повідомлень.
Користувачі системи	1) користувач (клієнт).
Вихідні дані	1) список доданих контактів; 2) надіслані та отримані текстові повідомлення через графічний інтерфейс у вигляді діалогу; 3) інформація про акаунт у розділі налаштувань.

ПРИЗНАЧЕННЯ ТА МЕЖІ ПРОЄКТУ

Призначення системи (застосунку), для якої розробляється програмне забезпечення

Призначенням застосунку є організація середовища для обміну тестовими повідомленнями.

Погодження, що ухвалені в програмній документації

Узгоджено, що для розробки ПЗ буде використовуватись PyQt [4].

Межі проєкту ПЗ

Крайня дата завершення роботи над ПЗ – 15.06.2024р.

ЗАГАЛЬНИЙ ОПИС

Сфера застосування

ПЗ не має обмежень щодо сфери застосування, так як налагоджена комунікація є однією з основних потреб суспільства, як міжособистісна так і ділова.

Характеристики користувачів

Користувач повинен мати стабільний доступ до мережі Інтернет, персональний комп’ютер з ОС Linux та базові периферійні пристрої до нього.

Загальна структура і склад системи

Система складається з наступних компонентів:

– клієнт;

– сервер;

– дві БД для клієнта (окремо для даних лаунчера й самого месенджера) та одна для сервера.

Загальні обмеження

Обмеження використання ПЗ – наявність ПК з ОС на базі Linux та підключення до мережі Інтернет.

ФУНКЦІЇ СИСТЕМИ МЕСЕНДЖЕРУ З ФУНКЦІЄЮ ЛОКАЛЬНОГО ЗБЕРЕЖЕННЯ ІСТОРІЇ ЛИСТУВАННЯ

Функція надсилання повідомлення

Опис функції

Функція надсилання повідомлення дозволяє користувачу відправити необхідну інформацію, у вигляді тексту або файлу, до бажаного отримувача.

Вхідна і вихідна інформація

Вхідна інформація – текстове повідомлення або файл;

вихідна інформація – успішно відправлене повідомлення буде відображатись у інтерфейсі користувача.

Функціональні вимоги

БД для передачі інформації між клієнтами через сервер та доступ до мережі Інтернет.

Функція додавання іншого користувача у контакти за нікнеймом

Опис функції

Функція додавання іншого користувача у контакти за нікнеймом дозволяє користувачу зберігти профіль іншого клієнта у списку контактів, щоб спростити процес комунікації.

Вхідна і вихідна інформація

Вхідна інформація – нійнейм іншого користувача;

вихідна інформація – відображення доданого користувача у списку контактів.

Функціональні вимоги

БД для зберігання даних про контакти та доступ до мережі Інтернет.

Функція резервного копіювання історії листування

Опис функції

Функція резервного копіювання історії листування дозволяє зберігти історію листування у вигляді файлу БД, задля забезпечення можливості в майбутньому відновити дані листування, так як історія листування не зберігається на сервері.

Вхідна і вихідна інформація

Вхідна інформація – історія листування користувача;

вихідна інформація – файл резервного копіювання.

Функціональні вимоги

БД для збереження історії листування та доступ до мережі Інтернет.

ВИМОГИ ДО ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Джерела і зміст вхідної інформації (даних)

Основним джерелом вхідної інформації в даному програмному забезпечення є користувач, а саме його текстові повідомлення або файли.

Нормативно-довідкова інформація (класифікатори, довідники тощо)

До даного пункту вимог немає.

Вимоги до способів організації, збереження та ведення інформації

Організація та збереження інформації здійснюється за допомогою SQLite[5] та SQLAlchemy [6].

ВИМОГИ ДО ТЕХНІЧНОГО ЗАБЕЗПЕЧЕННЯ

Вимоги до технічного забезпечення не є жорсткими. У користувача повинен бути персональний комп’ютер з ОС на базі ядра Linux та підключення до мережі Інтернет.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура програмної системи

Архітектура програмного забезпечення складається з клієнтської частини, серверної частини та БД для кожної з цих частин.

Системне програмне забезпечення

Для функціонування клієнтської та серверної частин застосунку використовується операційна система на базі Linux.

Мережне програмне забезпечення

Для забезпечення мережевої функціональності використовується операційна система на базі Linux.

Програмне забезпечення ведення інформаційної бази

В якості організації бази даних обрано СКБД SQLite.

Мова і технологія розробки ПЗ

Програмне забезпечення розроблятиметься з використанням мови Python [7] та фреймворку PyQt [8]. В якості редактора коду використовується Visual Studio Code.

ВИМОГИ ДО ЗОВНІШНІХ ІНТЕРФЕЙСІВ

Інтерфейс користувача

Графічний інтерфейс користувача має бути розроблений за основними стандартами UX та UI дизайну. Клієнтський застосунок має дві частини: лаунчер та месенджер. Перше вікно, яке бачить користувач це лаунчер, який напочатку пропонує вибір між реєстрацією або входом в акаунт, у результаті чого показує необхідну форму. При вдалому вході в акаунт відкривається вікно месенджеру, який логічно розділений на 3 вкладки: листування, контакти, налаштування. Така побудова інтерфейсу буде легкою для розуміння навіть новим користувачам.

Апаратний інтерфейс

Апаратним інтерфейсом є персональний комп’ютер користувача.

Програмний інтерфейс

PyQT – це оболонка на мові програмування Python для бібліотеки Qt. Бібліотека реалізована в Python-модулях й охоплює близько 1000 класів.

Комунікаційний протокол

Для мережевого з'єднання між клієнтом та сервером використовуються TCP/IP [9] з’єднання.

ВЛАСТИВОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Доступність

ПЗ є доступним для будь-якого користувача, за умовою наявності апаратного забезпечення та доступу до мережі Інтернет у користувача.

Супроводжуваність

Застосунок не потребує супроводжуваності.

Переносимість

Програмне забезпечення може працювати на будь-якій ОС на базі ядра Linux, яка підтримується розробниками цих систем на сьогодення.

Продуктивність

Продуктивність напряму залежить від швидкості підключення до мережі Інтернет.

Надійність

Дані, які надає користувач при реєстрації та обміну інформацією у процесі використання застосунку мають бути приватними. Користувач має доступ тільки до своїх даних та лише у випадку успішно пройденої авторизації у системі.

Безпека

Аутентифікація користувачів при обміні даними з сервером відбувається за допомогою токенів доступу, які отримуються клієнтом кожного разу при успішній авторизації в системі. Також після встановлення з’єднання між клієнтом та сервером дані передаються в зашифрованому вигляді.

1.3 Опис інтерфейсу користувача

Інтерфейс користувача є ключовим елементом в процесі розробки програмного забезпечення, оскільки він визначає спосіб взаємодії між користувачем та програмою. Для створення ефективного та зручного інтерфейсу було використано макети, або mock-up.

Проаналізувавши популярні месенджери, зокрема їх інтерфейс, були створені mock-up за допомогою програмного забезпечення «draw.io», яке є відкритим інструментом для створення діаграм та макетів онлайн.

Draw.io дозволяє користувачам легко створювати блок-схеми, розумові карти, діаграми баз даних, UML [10] діаграми та багато інших видів візуальних представлень. Це програмне забезпечення підтримує різноманітність шаблонів та макетів, які можуть бути корисними для різних професійних потреб.

Застосунок поділяється на два вікна, які заміняють один одного : лаунчер для реєстрації й авторизації та сам месенджер.

Вікно лаунчера (рис. 1.4) має 3 сторінки, які містять:

– меню для вибору між реєстрацією та входом в акаунт;

– форма для реєстрації акаунту;

– таблиця вже авторизованих профілів та форма для входу в зареєстрований акаунт.

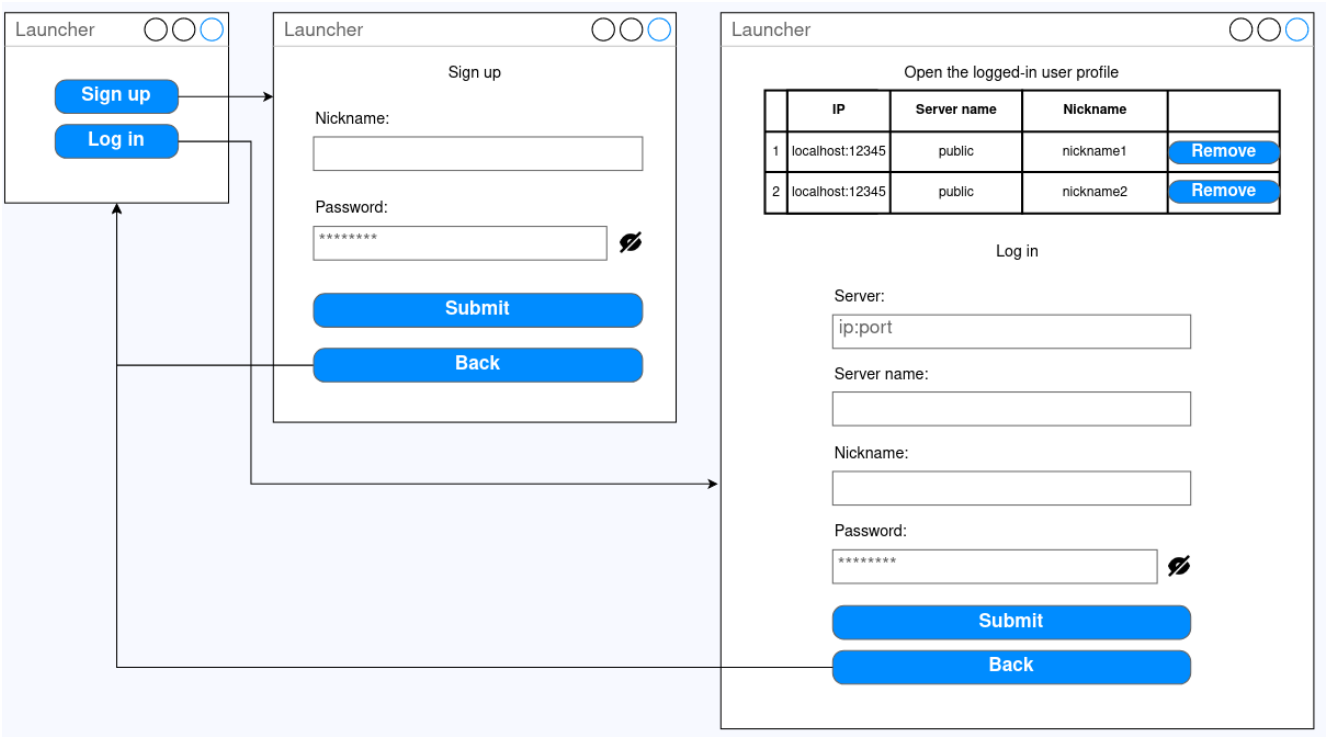

Рисунок 1.4 – Макет UI лаунчеру

На макеті UI вікна лаунчера також продемонстровано стрілками варіант переходу між сторінками, використовуючи відповідні кнопки.

На першій сторінці, при натисканні на кнопку "Sign up", головна сторінка вікна лаунчера буде змінена на сторінку реєстрації, при натисканні на кнопку "Log in" – зміна на сторінку авторизації. На обох сторінках, реєстрації та авторизації, є кнопка

"Back", котра поверне користувача назад до сторінки з меню.

Вікно месенджеру має 3 основні розділи: список листувань (вкладка "Messages"), список контактів (вкладка "Contacts"), налаштування профілю (вкладка "Settings"). Макет графічного інтерфейсу для вкладки "Messages" (рис. 1.5) наведено нижче.

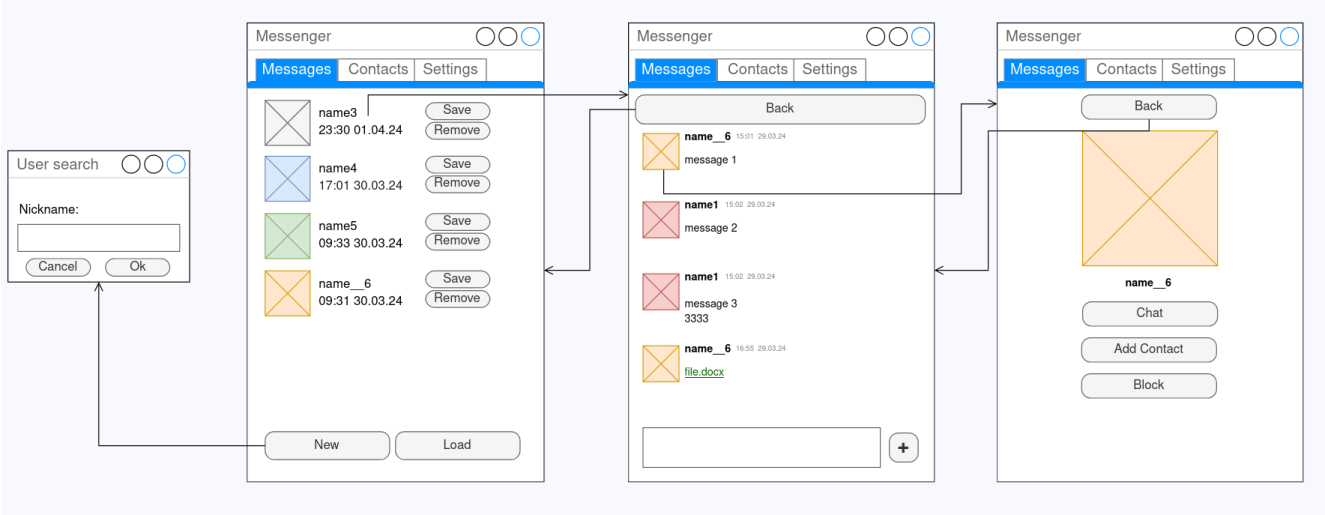

Рисунок 1.5 – Макет UI списку листувань

Макет графічного інтерфейсу списку листувань складається з трьох основних сторінок та одного додаткового вікна. За замовчуванням, головною сторінкою у вкладці "Messages" є сторінка, що містить перелік наявних листувань. Користувач може легко переходити між листуваннями, натискаючи на відповідний елемент списку, що відкриває сторінку з самим діалогом. В цьому діалозі доступна історія листування та функціонал відправлення нових повідомлень. Натиснувши на зображення профілю користувача, відкривається сторінка для перегляду профілю цієї особи, що надає більш детальну інформацію.

Додатково, в інтерфейсі присутнє вікно для пошуку користувачів у системі за їх нікнеймом. Це вікно відкривається при натисканні кнопки "New" на головній сторінці, що відповідає за створення нового діалогу. Цей функціонал спрощує процес пошуку та спілкування з іншими користувачами у системі, забезпечуючи зручність та швидкість взаємодії з програмою.

Макет графічного інтерфейсу для вкладки "Contacts" (рис. 1.6) представлено нижче.

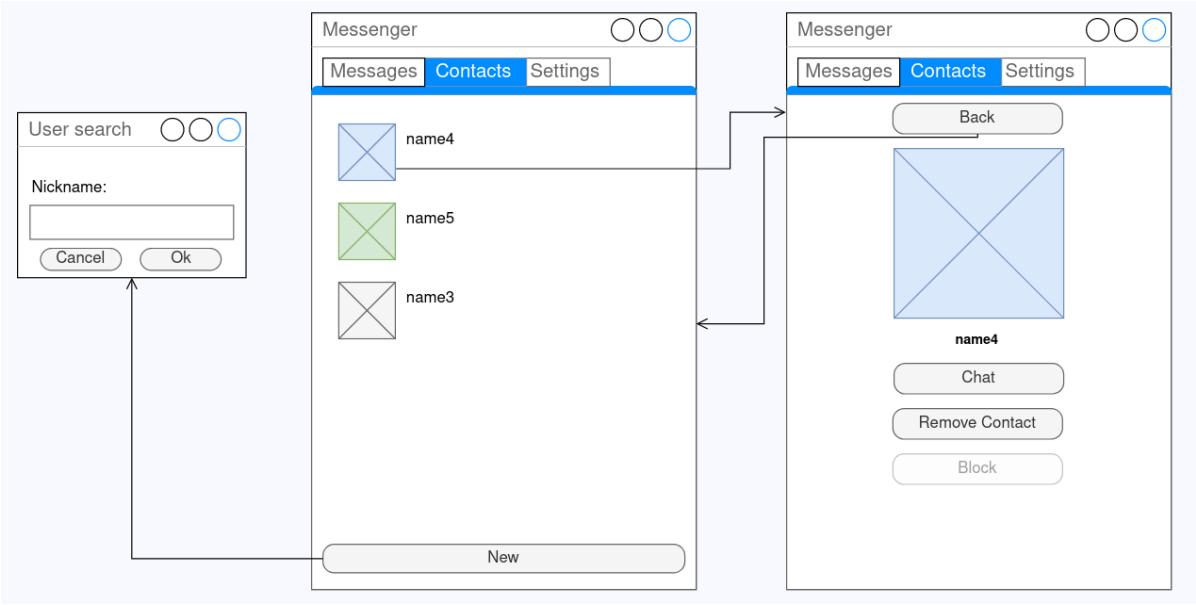

Рисунок 1.6 – Макет UI списку контактів

Змодельований UI списку контактів має, так само як і макет списку листувань, додаткове вікно для пошуку інших клієнтів, тільки вже з метою додавання у перелік контактів. Головна сторінка має список доданих контактів, при натисканні на котрі відображення змінюється на перегляд профілю контакту.

Нижче наведено макет UI для вкладки "Settings" (рис. 1.7).

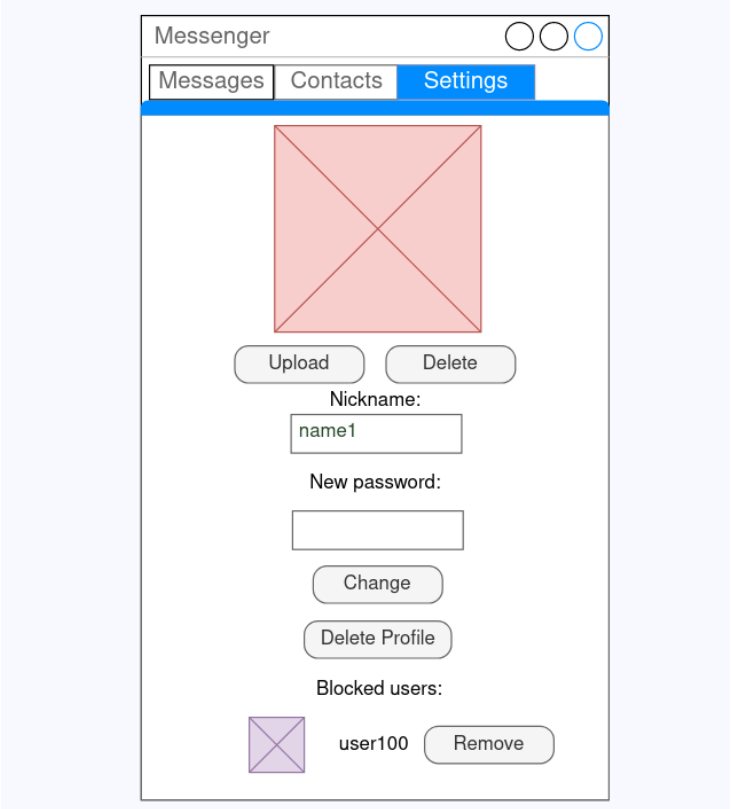

Рисунок 1.7 – Макет UI налаштувань

Макет інтерфейсу вкладки налаштувань містить одну сторінку, де користувач може переглянути та змінити інформацію свого профілю, включаючи особисті дані. Крім того, користувач має можливість видалити свій профіль, якщо це стає необхідним. Ще однією доступною можливістю на цій сторінці є керування списком заблокованих користувачів, яке здійснюється за допомогою кнопки "Remove".

Створені макети, є важливими етапами для успішної реалізації застосунку. Вони допомагають сформувати загальне уявлення та бачення майбутньої системи, спрощуючи процес розробки та забезпечуючи відповідність вимогам користувачів. Такий підхід дозволяє ефективно впроваджувати новий продукт і забезпечує його успішне функціонування.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи бакалавра був проведений детальний аналіз наявних аналогів системи, що розробляється, також виконавши їх порівняння між собою та зосереджуючи увагу на перевагах та недоліках кожного. Це дозволило не лише з’ясувати, які функції та можливості вже існують на ринку, але й спростило процес планування переліку основних задач, яке повинно виконувати програмне забезпечення.

Також створено специфікацію вимог до ПЗ, що містить детальний опис призначення та обмежень проєкту, загальний опис проєкту та його головних функцій, вимог до інформаційного та технічного забезпечення, інтерфейсів, а також властивостей програми.

Надані макети графічного інтерфейсу користувача та докладні описи до них для системи, що розробляється, є важливим етапом, який полегшить майбутню розробку застосунку. Ці макети дозволять чітко уявити, як буде виглядати та функціонувати програма, що сприятиме ефективнішому процесу розробки та забезпечить високу якість кінцевого результату.

2 МОДЕЛЮВАННЯ ЗАСТОСУНКУ

2.1 Сценарії використання

Нижче наведено основні повні сценарії роботи системи, зокрема створення профілю у системі (див. табл. 2.1), вхід до системи (див. табл. 2.2), створення діалогу (див. табл. 2.3), відправка повідомлення (див. табл. 2.4). Будуть описані саме такі сценарії, бо створення профілю та вхід є ключовим для того, щоб успішно використовувати головний функціонал ПЗ, а відправка повідомлень відображає основну мету існування месенджерів.

Таблиця 2.1 – Сценарій використання 1

Usecase section	**Comment**
Use Case Name	Створити профіль у системі
Scope	Месенджер
Level	Успішно пройти процес реєстрації.
Primary Actor	Клієнт
Stakeholders and interests	1) клієнт – отримати профіль для користування системою.
Preconditions	Клієнт повинен мати: встановлене ПЗ, незареєстровані дані для реєстрації.
Success guarantee	1) клієнт повинен мати встановлене ПЗ; 2) клієнт має використовувати стабільне підключення до мережі Інтернет; 3) клієнт має вводити коректні дані для створення профілю (нікнейм, пароль).
Main Success Scenario	1) клієнт відкриває ПЗ з формою реєстрації та вводить унікальний нікнейм та пароль у спеціалізовані поля, підтверджує форму; 2) система реєструє клієнта та відкриває доступ до користування ПЗ.
Extensions	а) клієнт не вводить дані: 1) клієнт відкриває застосунок з формою для реєстрації; 2) клієнт залишає поля вводу даних незаповненими; 3) клієнт натискає кнопку підтвердження;

Кінець таблиці 2.1

Extensions	4) система видає помилку валідації та не реєструє клієнта; а) клієнт вводить некоректні дані: 1) клієнт відкриває застосунок з формою для реєстрації; 2) клієнт вводить некоректні дані; 3) клієнт натискає кнопку підтвердження; 4) система видає помилку валідації та не реєструє клієнта; б) клієнт вводить всі дані, але нікнейм не унікальний: 1) клієнт відкриває застосунок з формою для реєстрації; 2) клієнт вводить дані, але нікнейм не унікальний; 3) клієнт натискає кнопку підтвердження; 4) система видає помилку та не реєструє клієнта.
Special Requirements	Швидкість доступу до мережі Інтернет клієнта має бути більшою ніж 60 кб/сек.
Frequency of Occurrence	Система здатна функціонувати практично без перерв.

Таблиця 2.2 – Сценарій використання 2

Usecase section	**Comment**
Use Case Name	Увійти до системи
Scope	Месенджер
Level	Успішно пройти процес авторизації в месенджері.
Primary Actor	Клієнт
Stakeholders and interests	1) клієнт – отримати доступ до функціоналу застосунку.
Preconditions	Клієнт повинен мати: встановлене ПЗ, дані зарєстрованого профілю.
Success guarantee	1) клієнт повинен мати встановлене ПЗ; 2) клієнт має використовувати стабільне підключення до мережі Інтернет; 3) клієнт має вводити коректні дані для авторизації (дані сервера, нікнейм, пароль).

Кінець таблиці 2.2

Main Success Scenario	1) клієнт відкриває ПЗ з формою авторизації 2) вводить коректні дані сервера, ніку та паролю у спеціалізовані поля; 3) система авторизує клієнта та відкриває доступ до користування ПЗ.
Extensions	а) клієнт не вводить дані: 1) клієнт відкриває застосунок з формою авторизації; 2) клієнт залишає поля вводу даних незаповненими; 3) клієнт натискає кнопку підтвердження; 4) система видає помилку та не авторизує клієнта; б) клієнт вводить недійсні дані: 1) клієнт відкриває застосунок з формою авторизації; 2) клієнт вводить недійсні дані; 3) клієнт натискає кнопку підтвердження; 4) система видає помилку та не авторизує клієнта.
Special Requirements	Швидкість доступу до мережі Інтернет клієнта має бути більшою ніж 60 кб/сек.
Frequency of Occurrence	Система здатна функціонувати практично без перерв.

Таблиця 2.3 – Сценарій використання

Usecase section	**Comment**
Use Case Name	Створити діалог
Scope	Месенджер
Level	Успішно створити діалог з іншим клієнтом.
Primary Actor	Клієнт
Stakeholders and interests	1) клієнт – отримати можливість обміну повідомленнями, файлами.
Preconditions	Клієнт повинен мати: встановлене ПЗ з пройденою авторизацією, іншого клієнта у своїх контактах або знати його нікнейм.

Кінець таблиці 2.3

Success guarantee	1) клієнт повинен мати встановлене ПЗ з пройденою авторизацією; 2) клієнт має використовувати стабільне підключення до мережі Інтернет; 3) клієнт має потрібний профіль у контактах або знає його нікнейм.
Main Success Scenario	1) клієнт відкриває ПЗ з авторизованим профілем; 2) клієнт переходить у розділ списку контактів; 3) клієнт обирає необхідного користувача або вводить його нікнейм в потрібне поле, після натискання на кнопку "New"; 4) у клієнта автоматично створюється діалог з відповідним клієнтом.
Extensions	а) клієнт не має потрібний профіль у контактах та не знає нікнейму іншого клієнта: 1) клієнт переходить у розділ списку контактів; 2) клієнт не знаходить у списку потрібний профіль або не вводить нікнейм у спеціальне поле, після натискання на кнопку "New"; 3) діалог не створюється; б) клієнт не має потрібний профіль у контактах та вводить недійсний нікнейм іншого клієнта: 1) клієнт переходить у розділ списку контактів; 2) клієнт не знаходить у списку потрібний профіль та вводить недійсний нікнейм у спеціальне поле, після натискання на кнопку "New"; 3) система сповіщує про помилку, діалог не створюється.
Special Requirements	Швидкість доступу до мережі Інтернет клієнта має бути більшою ніж 60 кб/сек.
Frequency of Occurrence	Система здатна функціонувати практично без перерв.

Таблиця 2.4 – Сценарій використання 4

Usecase section	**Comment**
Use Case Name	Написати повідомлення
Scope	Месенджер
Level	Успішно написати повідомлення іншому клієнта.
Primary Actor	Клієнт
Stakeholders and interests	1. клієнт – надсилання повідомлень, файлів.
Preconditions	Клієнт повинен мати: встановлене ПЗ з пройденою авторизацією, створений діалог з іншим клієнтом.
Success guarantee	1) клієнт повинен мати встановлене ПЗ з пройденою авторизацією; 2) клієнт має використовувати стабільне підключення до мережі Інтернет; 3) клієнт має наявний діалог.
Main Success Scenario	1) клієнт відкриває ПЗ з авторизованим профілем; 2) клієнт обирає необхідний діалог зі списку листувань; 3) клієнт відправляє текстове повідомлення або файл через меню прикріплення файлу.
Extensions	а) клієнт відправляє пусте повідомлення: 1) клієнт обирає необхідний діалог зі списку листувань; 2) клієнт залишає поле вводу повідомлення пустим та не прикріплює жодного файлу, натискає кнопку підтвердження; 3) повідомлення не відправляється; б) клієнт намагається відправити файл розміром більше 200Мб: 1) клієнт обирає необхідний діалог зі списку листувань; 2) клієнт намагається прікріпити файл розміром більше 200Мб; 3) система не дозволяє прикріпити такий файл, тому повідомлення не відправляється.

Кінець таблиці 2.4

Special Requirements	Швидкість доступу до мережі Інтернет клієнта має бути більшою ніж 60 кб/сек.
Frequency of Occurrence	Система здатна функціонувати практично без перерв.

Створені повні сценарії використання відіграють велику роль у подальшій розробці системи, а головне зрозуміти потреби користувача, як вони будуть взаємодіяти з системою, які функції використовують та які вимоги й потреби вони мають.

2.2 Діаграма варіантів використання

Окрім сценаріїв використання, які більше описують конкретні взаємодії між акторами та системою у вигляді послідовності кроків, необхідно створити діаграму використання.

Діаграма варіантів використання (use case diagram) – це тип діаграми в рамках мови моделювання UML, яка використовується для візуалізації взаємодії між користувачами (акторами) та системою, шляхом ідентифікації та опису різних варіантів використання (прецедентів).

Окрім опису конкретних сценаріїв використання, які в подробицях описують послідовність дій, діаграма варіантів використання фокусується на відносинах між цими сценаріями та акторами.

Діаграма варіантів використання може містити такі елементи:

– актори, які представляють зовнішніх користувачів або інші системи, які взаємодіють з системою;

– прецеденти, котрі представляють різні варіанти використання системи, тобто функції або можливості, які надає система своїм акторам;

– відносини між акторами та прецедентами, які показують, як актори взаємодіють з прецедентами.

Ця діаграма (рис. 2.1) допомагає зрозуміти, які функції надає система та які актори взаємодіють з цими функціями. Це відображає структуру системи та дозволяє визначити основні варіанти використання системи з точки зору її користувачів.

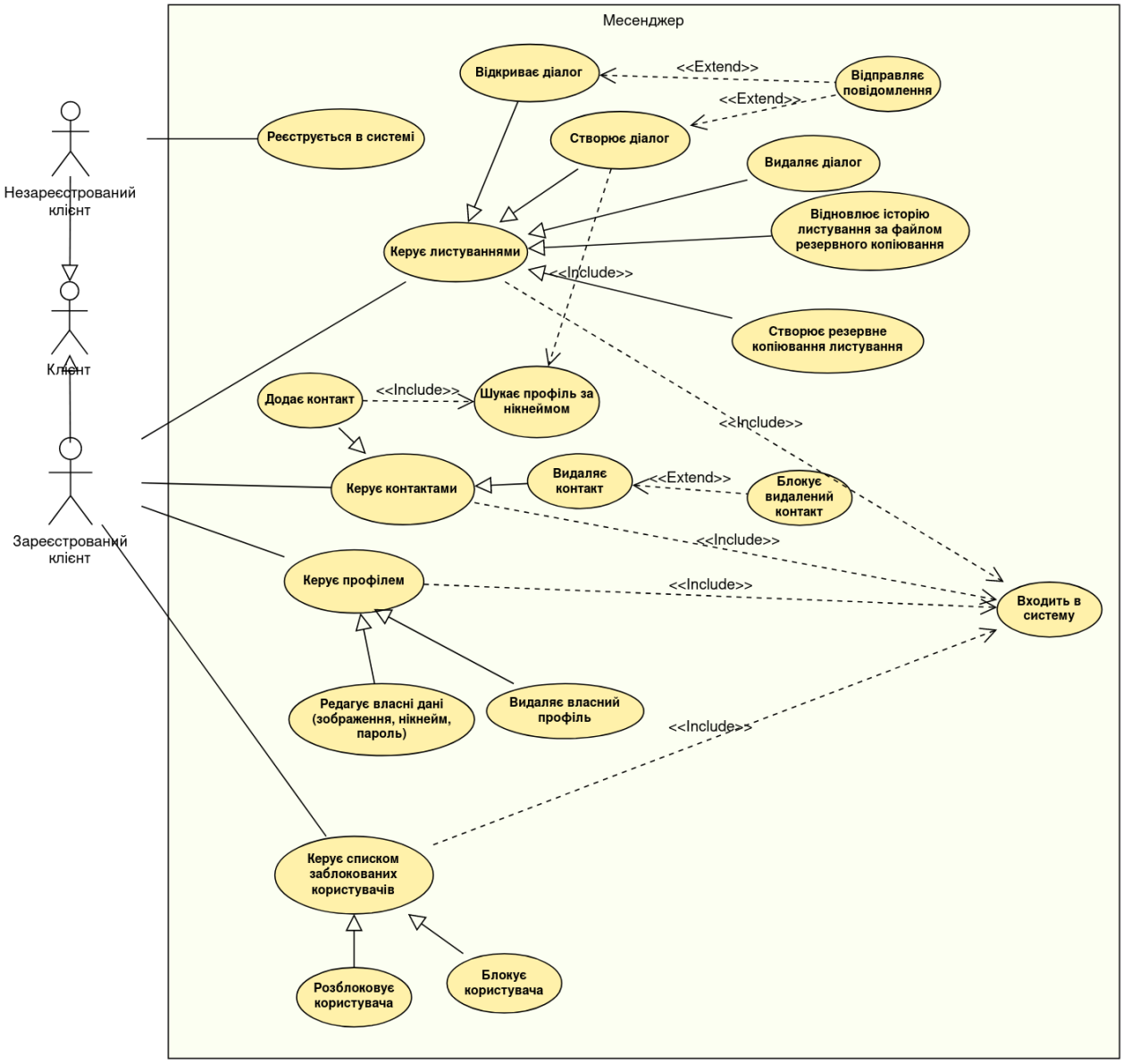

Рисунок 2.1 – Діаграма варіантів використання

На цій діаграмі показано 3 актори, які представляють собою варіанти одного клієнта. Існує абстрактний актор "Клієнт", який створений щоб показати, що зареєстрований і незареєстрований клієнт це просто дві різні форми клієнта, але притому зареєстрований клієнт вже немає функціоналу реєстрації в системі. Для працездатності більшості функціоналу є необхідним вхід до системи, що показано відношенням включення ("include"). Також на діаграмі присутнє відношення

розширення ("extend"), яке визначає такий тип відношення, коли один варіант за певних умов повністю використовує інший, розширюючи функціонал.

2.3 Діаграми діяльності

Щоб наглядно показати послідовність дій або процесів майбутнього програмного забезпечення, використовують діаграми діяльності. Вони допомагають розуміти логіку програми та взаємодію об'єктів чи компонентів, та включають в себе такі основні елементи:

– дії ("actions") це конкретні кроки або операції, які виконуються в процесі виконання програми;

– рішення ("decisions") це умовні переходи, що визначають, які дії будуть виконуватися в залежності від умови;

– паралельність ("concurrency"), яка відображає як декілька дій можуть відбуватися паралельно;

– петлі ("loops"), повторні виконання певних дій, поки виконується певна умова;

– старт та фініш ("start and end points"), які позначають початок та завершення послідовності дій.

Діаграми діяльності використовуються для моделювання різних процесів, починаючи від керування бізнес-процесами до опису алгоритмів програм. Їхній графічний характер робить їх легкими для розуміння і сприйняття командою розробників.

Діаграми діяльності в певній мірі є універсальними, так як можуть бути використані на різних етапах розробки ПЗ, починаючи від аналізу та проєктування до реалізації й тестування.

На діаграмах діяльності входу в профіль (рис. 2.2), діяльності відправки повідомлення (рис. 2.3), діяльності редагування налаштувань (рис. 2.4) було використано такі елементи: дії, рішення, старт та фініш.

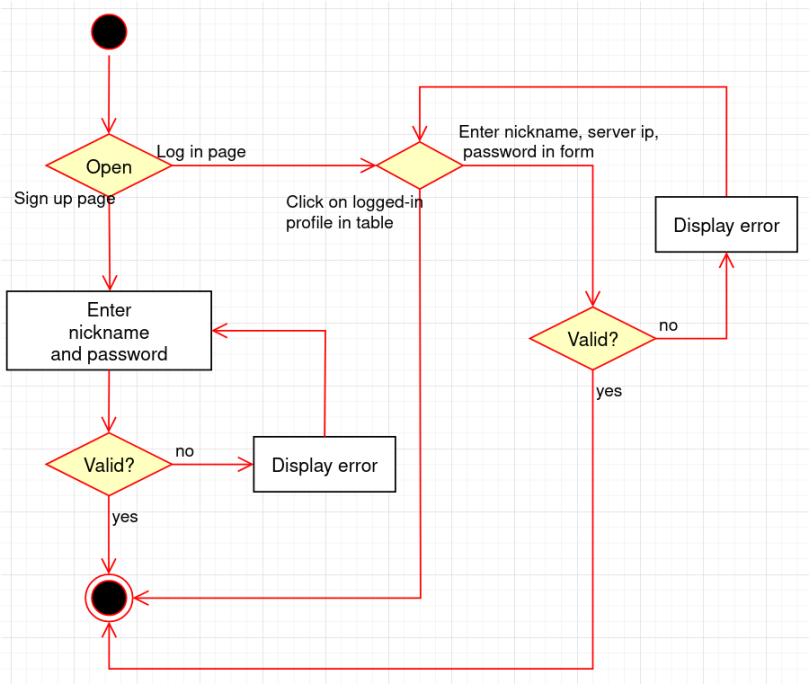

Рисунок 2.2 – Діаграма діяльності входу в профіль

Діаграма діяльності входу в профіль починається з рішення яку сторінку вікна лаунчеру відкрити. Вибір буде залежати від того, чи користувач хоче зареєструватись чи авторизуватись.

З реєстрацією подальший процес простий, користувач повинен ввести дані, і якщо вони невалідні, то користувачу показує вікно помилки. Невалідними вважаються дані, котрі не входять в певний інтервал кількості символів, або повинні містити тільки обмежені типи символів, в деяких випадках використання всіх може бути обов’язковим. Наприклад, для валідації паролю треба щоб дані містили хоча б одну велику літеру, маленьку літеру та цифру. Навіть за умови що дані відповідають цим умовам, може статись таке, що обраний нікнейм для реєстрації вже зайнятий, і тоді сервер передасть саме цю помилку, а інтерфейс користувача її обробить виводом вікна з сповіщенням про помилку. Після кожної помилки основне вікно не змінюється, щоб користувач мав змогу змінити дані на валідні та продовжити користування застосунком.

При виборі на початку входу в акаунт, інтерфейс користувача відобразить таблицю з збереженими авторизованими користувачем в минулому профілями та формою для нового входу. Тому клієнта системи знову надаються варіанти рішення.

Щоб зайти в профіль з таблиці, треба на неї двічі натиснути, і якщо файл БД та

інші дані цього користувача не пошкоджені в системі, то здійснеться успішний вхід. У випадку нового входу, також виконується валідація даних так само, як в розділі реєстрації, але додатково ще перевіряється правильність формату даних адреси сервера, на якому був зареєстрований профіль. Може статись так, що коли користувач вводить дані входу, які вже збережені в таблиці авторизованих профілів. В такому варіанті система просто підтягне дані з таблиці і не буде виконувати запит на сервер для авторизації. В інших випадках, після успішної авторизації на сервері, в локальній БД лаунчеру буде додана інформація про нового користувача, яка буде відображена в таблиці. А також створяться файли БД месенджеру для цього користувача, для збереження даних листування, налаштувань профілю, списку контактів та заблокованих користувачів. Цей файл бази даних підтягується при кожному успішному вході в акаунт.

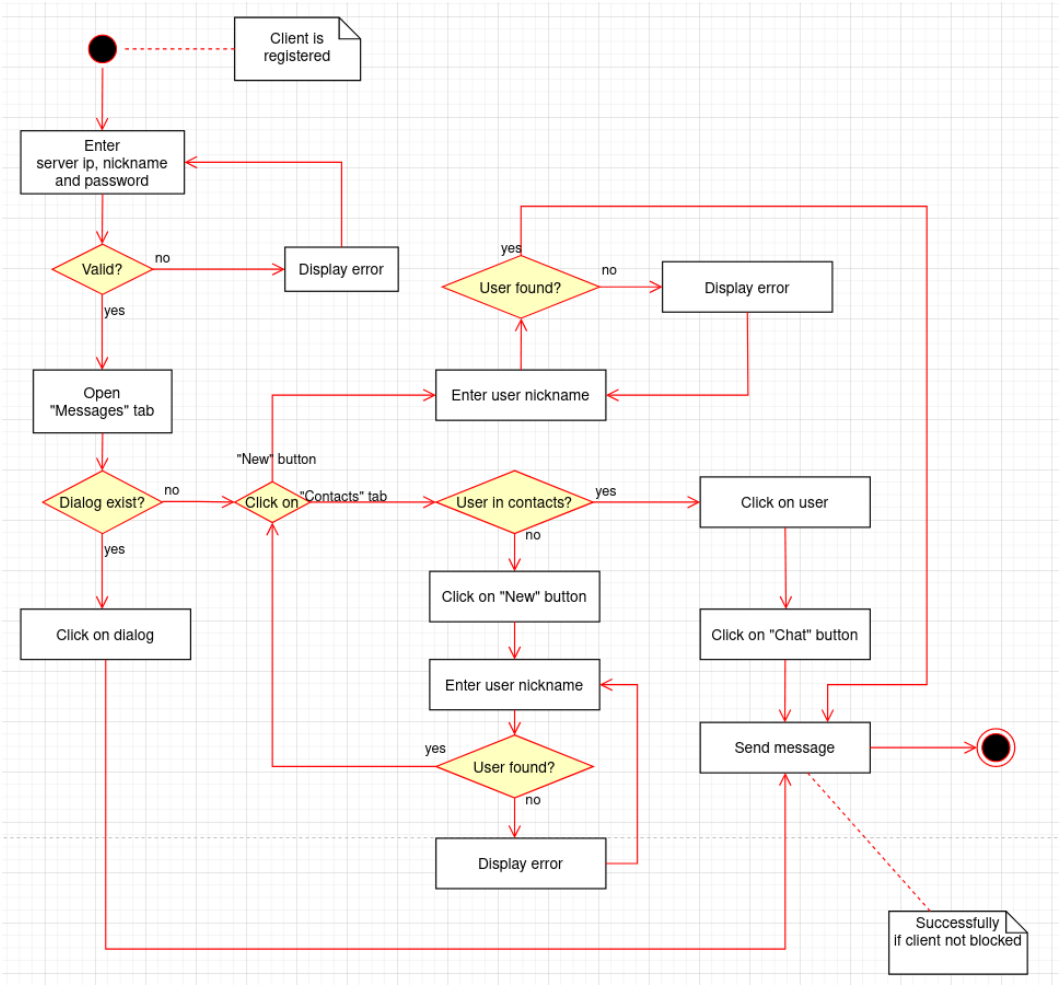

Рисунок 2.3 – Діаграма діяльності відправки повідомлення

Діаграма діяльності відправки повідомлення передбачає, що клієнт вже зареєстрований в системі.

Щоб досягнути мети відправлення повідомлення, першим кроком буде заповнити форму для входу в акаунт, за умови що користувач не має ще даних для швидкого входу у таблиці збережених авторизованих профілів.

Як і в попередній діаграмі, графічний інтерфейс користувача валідує дані. При успішному вході в систему клієнта відкривається вікно месенджера, яке містить віджет вкладок, який організовує всі сторінки месенджеру. Цей віджет складається з панелі вкладок та "область сторінки", яка використовується для відображення сторінок, пов’язаних з панеллю вкладок. За замовчуванням, при вході в систему та відкритті месенджеру відображається сторінка зі списком листувань.

Щоб написати повідомлення, користувач має звернути увагу, чи існує потрібний йому діалог. Якщо діалог вже існує, то залишається тільки натиснути на нього, написати повідомлення та відправити його.

В разі коли потрібного діалогу немає, клієнта надається вибір: натиснути кнопку "New" на сторінці списку листувань або перейти на вкладу зі списком контактів. Перше рішення буде швидким, бо користувачу треба тільки вписати нікнейм бажаного клієнта для комунікації та почати листування.

При рішенні перейти на іншу вкладку, сторінка змінюється, і тепер користувач бачить список своїх контактів, кожному з котрих він має можливість написати. Якщо користувач, котрому планував написати клієнт є у списку, то він просто натискає на цей рядок. Це призведе до перегляду профілю контакту, на якому буде кнопка "Chat", яка відкриває діалог.

Якщо певного користувача у списку немає, то подальші дії схожі як при створенні нового діалогу на вкладці списку листувань, а саме відкривається вікно для вводу нікнейму, і якщо такий користувач знайдений, то автоматично створиться порожній діалог.

У кожному з варіантів, передбачається що клієнт не є у списку заблокованих користувачів у налаштуваннях інших клієнтів.

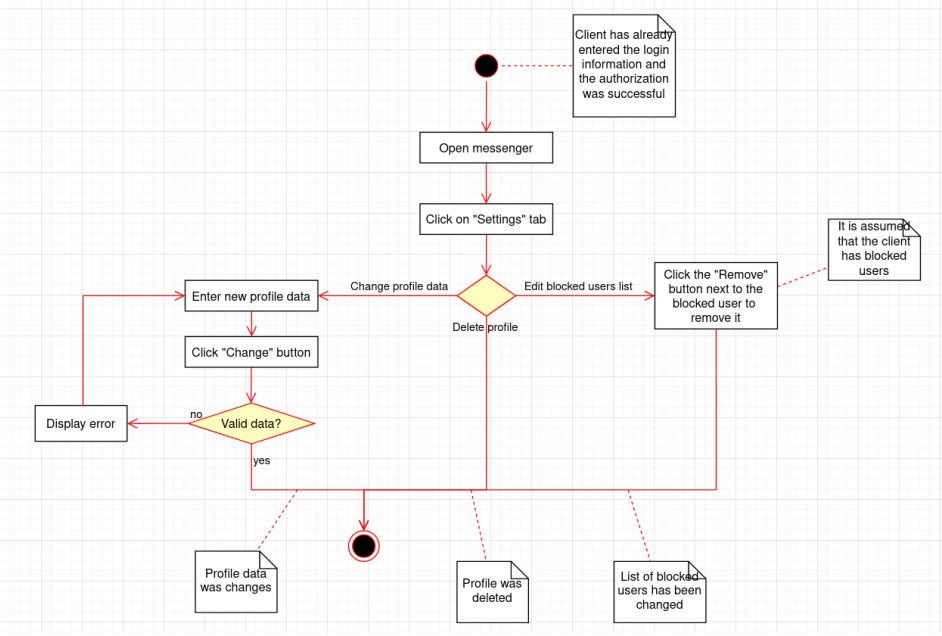

Рисунок 2.4 – Діаграма діяльності редагування налаштувань

Діаграма діяльності редагування налаштувань демонструє функціонал, який представляють такі рішення: змінити інформацію профілю, видалити профіль, видалити інших клієнтів зі списку заблокованих користувачів. При зміні даних профілю система проводить валідацію, видалення профілю можливе лише за умови що профіль зареєстрований на сервері, редагування списку заблокованих користувачів потребує наявність хоча б одного клієнта в цьому списку.

2.4 Діаграма розгортання

Діаграма розгортання – це один з видів діаграм в рамках мови моделювання UML, яка використовується для візуалізації фізичної архітектури системи, включаючи розташування компонентів та їх зв'язки.

На діаграмі (рис. 2.5) є два основних вузли: Клієнт ("Client") та Сервер ("Server"). Між клієнтом та сервером встановлено зв'язок за допомогою протоколу TCP/IP, що позначено на діаграмі відповідним зв'язком.

Клієнтська частина містить в собі:

– клієнтський застосунок, який надає графічний інтерфейс та обробку взаємодії з цим інтерфейсом, використовуючи сигнали та слоти, у результаті чого надсилаючи запити на сервер;

– БД лаунчеру, яка відповідає за збереження даних авторизованих користувачів, які частково будуть відображатись для таблиці збережених авторизованих профілів у вікні входу;

– БД месенджеру, яка відповідає за збереження всієї необхідної інформації активного користувача системи для відображення сторінок месенджеру.

Серверна частина містить в собі:

– серверний застосунок, який оброблює запити, отримані зі сторони клієнта;

– базу даних, яка зберігає всю необхідну для працездатності системи, інформацію про наявних користувачів, але не зберігаючи історію листувань.

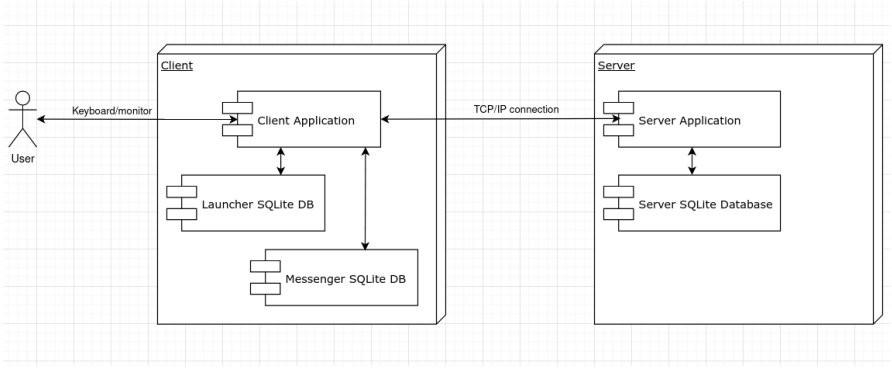

Рисунок 2.5 – Діаграма розгортання

Створена діаграма розгортання UML допомагає чітко відобразити взаємозв’язки між компонентами системи та їх фізичне розташування, що спрощує розуміння архітектури та процесу розгортання.

Висновки до розділу 2

Другий розділ кваліфікаційної роботи бакалавра присвячений моделюванню месенджеру з функцією локального збереження історії листування. За допомогою розробленої діаграми використання продемонстровано основні сценарії роботи системи. Побудовано діаграми діяльності для надання уявлення про роботу алгоритмів функціонування застосунку. Також створено діаграму розгортання для загального розуміння архітектури системи месенджеру.

3 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

3.1 Діаграми класів, компонентів та пакетів

В попередньому розділі було розглянуто такі UML-діаграми: варіантів використання, діяльності, розгортання. Для проєктування системи застосунку також необхідно створити діаграми класів, компонентів, пакетів.

Діаграми класів

Для моделювання статичної структури системи можна використати діаграми класів, які є фундаментальними для об'єктно-орієнтованого проєктування. Ця діаграма відображає класи системи, їх атрибути, методи та взаємозв'язки (асоціації, агрегації, композиції, наслідування) між ними. Це допоможе зрозуміти, як дані організовані і як взаємодіють різні компоненти системи. Діаграма класів сприяє створенню зрозумілої та узгодженої структури коду, що полегшує подальше розширення та обслуговування системи. Класи сервера, атрибути класів та методи у вигляді діаграми (рис. 3.1) наведено нижче.

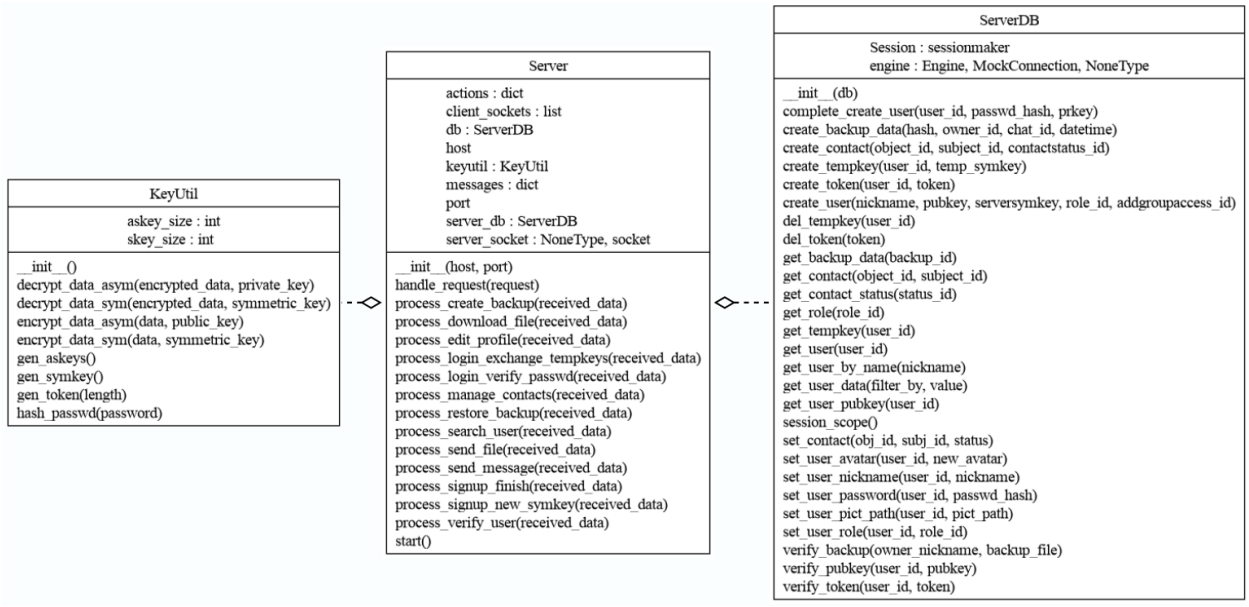

Рисунок 3.1 – Діаграма класів сервера для реєстрації та авторизації

Створено структуру класів сервера, яка забезпечує повний цикл обробки запитів від клієнтів, включаючи реєстрацію, авторизацію, управління контактами, обмін повідомленнями та файлами, створення і відновлення резервних копій, а також

редагування профілю користувача. Для цього було використано 3 класи: "Server", "ServerDB", "KeyUtil".

Клас "Server" відповідає за запуск сервера, обробку з'єднань та обмін даними з клієнтами. Цей клас використовує можливості класу "ServerDB", який забезпечує взаємодію з базою даних, включаючи створення, збереження, отримання та видалення даних користувачів, повідомлень, файлів, контактів та резервних копій, використовуючи SQLAlchemy для ORM і SQLite як СКБД. Клас "KeyUtil" надає функції для генерації та управління криптографічними ключами, шифрування та дешифрування даних, а також забезпечує безпеку обміну інформацією між клієнтами та сервером.

Нижче наведено класи клієнта (рис. 3.2), їх атрибути та методи, які для прикладу відповідають за функціонал доступний у лаунчері.

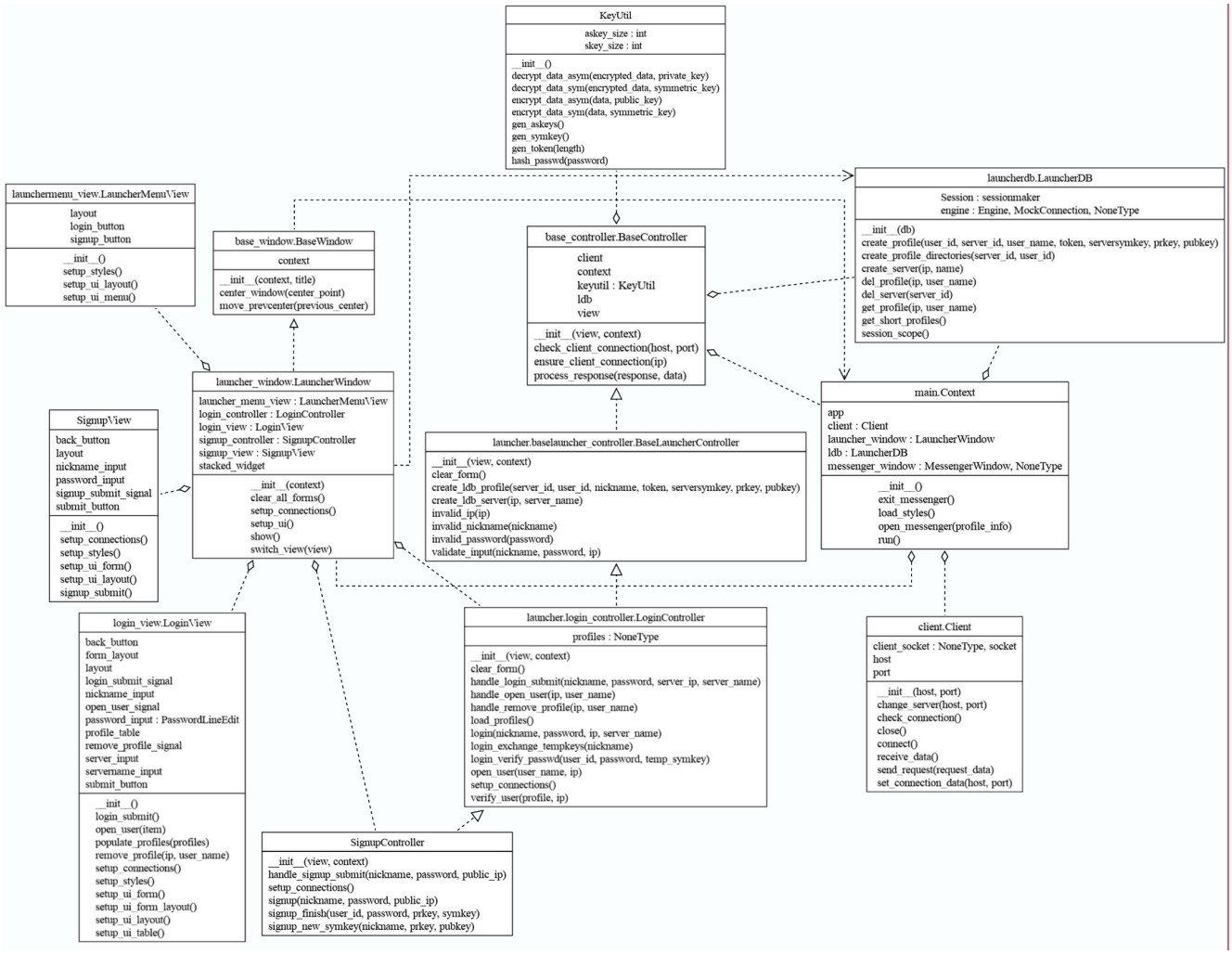

Рисунок 3.2 – Діаграма класів клієнта для вікна лаунчера

Клас "Context" є головним, саме він запускає весь застосунок, ініціалізує з’єднання з сервером, базою даних лаунчеру та відкриває його вікно, керує зміною вікон між лаунчером та месенджером.

Клас "LauncherDB" відповідає за операціями над БД лаунчеру.

Клас "Client" відповідає за підключення до сервера, обмін даними з ним за допомогою сокетів, відправку запитів і отримання відповідей, а також управління станом з'єднання.

Клас "KeyUtil" використовується у класах контролерів, для генерації ключів шифрування, для шифрування та дешифрування інформації, хешування паролю.

Класи "BaseWindow", "LauncherWindow" відповідають за створення вікон застосунку. Класи "SignupView", "LoginView", "LauncherMenuView" існують для створення відображення всередині вікон GUI.

Класи "BaseController", "BaseLauncherController", "LoginController", "SignupController" – це контролери клієнтського застосунку, котрі здебільшого займаються обробкою взаємодії користувача з інтерфейсом.

Діаграма компонентів

Діаграма компонентів фокусується на фізичному аспекті системи, моделюючи її архітектуру у вигляді взаємодіючих компонентів. Вона показує, як різні частини системи взаємодіють одна з одною та які інтерфейси використовуються для комунікації між ними.

Діаграма компонентів дозволяє візуалізувати, як програмні елементи системи розподілені та взаємопов'язані, забезпечуючи таким чином розуміння архітектури системи на високому рівні.

Ця діаграма є ключовою для забезпечення розподіленості та модульності системи, оскільки вона допомагає виявити залежності між компонентами та планувати інтеграцію системи. Завдяки їй можна чітко визначити архітектурні рішення, які найкраще підходять для побудови масштабованих і підтримуваних систем. Побудована діаграма компонентів клієнтського застосунку показана нижче (рис. 3.3).

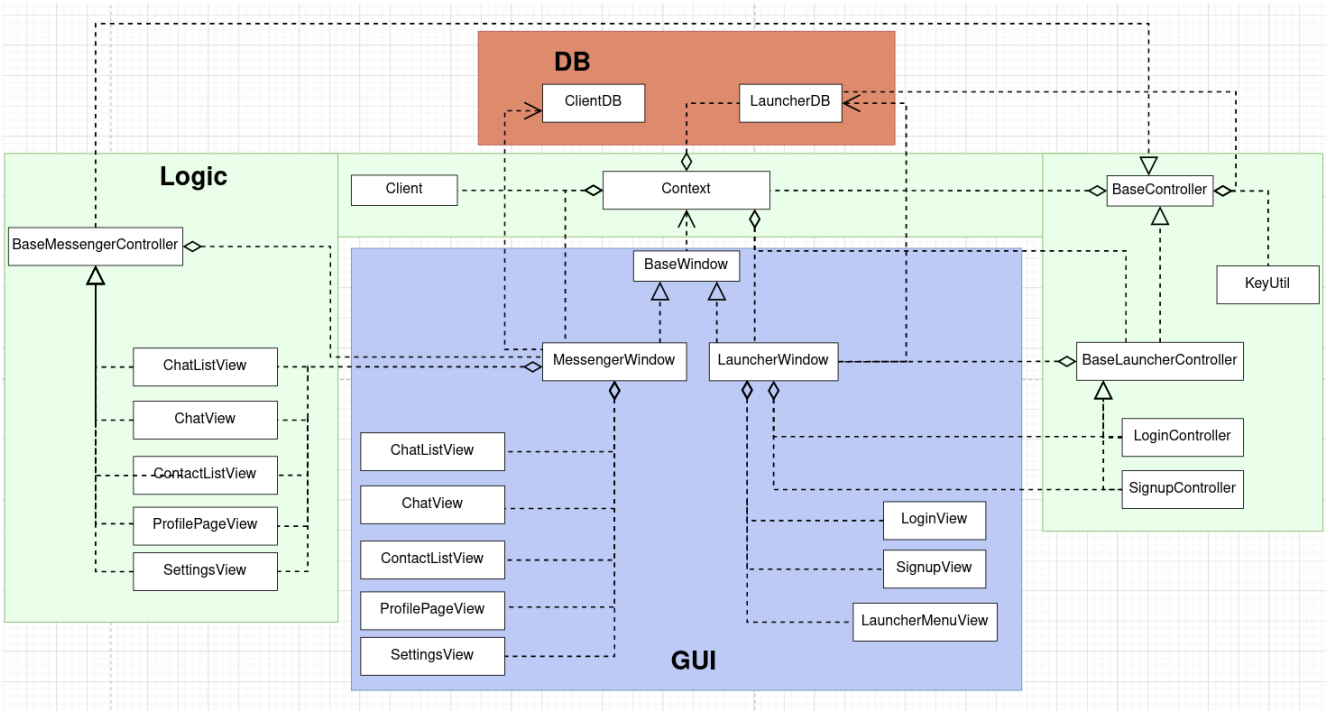

Рисунок 3.3 – Діаграма компонентів

На діаграмі продемонстровано 3 головні компоненти: "DB" – відповідає за функціонал роботи з базою даних; "Logic" – містить в собі класи, котрі відповідають за основну логіку програми, обробку взаємодії користувача з GUI; "GUI" – відповідає за відмалювання інтерфейсу користувача.

Діаграма пакетів

Для організації елементів моделі у групи, що полегшує їх управління та навігацію, можна створити діаграму пакетів. Вона показує, як класи, інтерфейси та інші елементи організовані у пакети та які залежності існують між цими пакетами. Це особливо корисно для великих застосунків, де важливо підтримувати зрозумілу ієрархію та структуру коду.

Діаграма пакетів допомагає побачити загальну структуру системи на високому рівні, забезпечує модульність та знижує складність управління кодом.

Нижче наведено такі діаграми пакетів для клієнтської частини застосунку (рис. 3.4) та серверної (рис. 3.5).

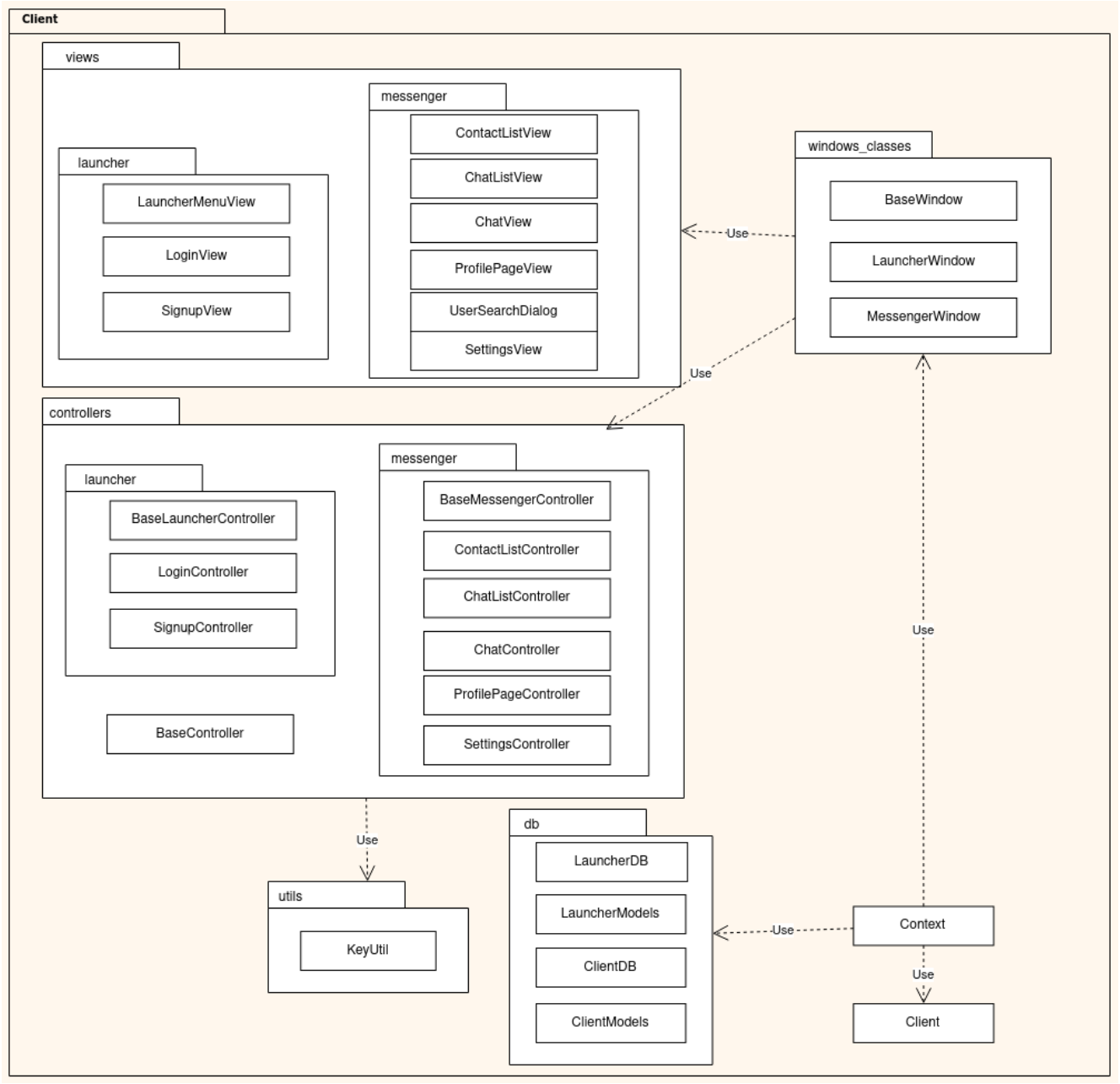

Рисунок 3.4 – Діаграма пакетів клієнта

Ця діаграма пакетів клієнта має 5 пакетів вищого рівню та 2 звичайних компонента. Пакети "views" та "controllers" мають в собі по 2 пакети "launcher" та "messenger", щоб розділити відображення з контролерами на те що пов’язано з лаунчером та месенджером. Також пакет "controllers" має в собі простий компонент BaseController та залежність від пакету "utils".

Пакет "windows_classes" відповідає за створення вікон, тому має залежність як і від "controllers" так і від "views".

Простий компонент "Context" є головним для застосунку клієнта, тому він має

відношення залежності до пакету "db", "windows_classes", та компоненту Client, які в свою чергу відповідають за керування клієнтською БД, створення вікон, та налаштування зв’язку з сервером відповідно.

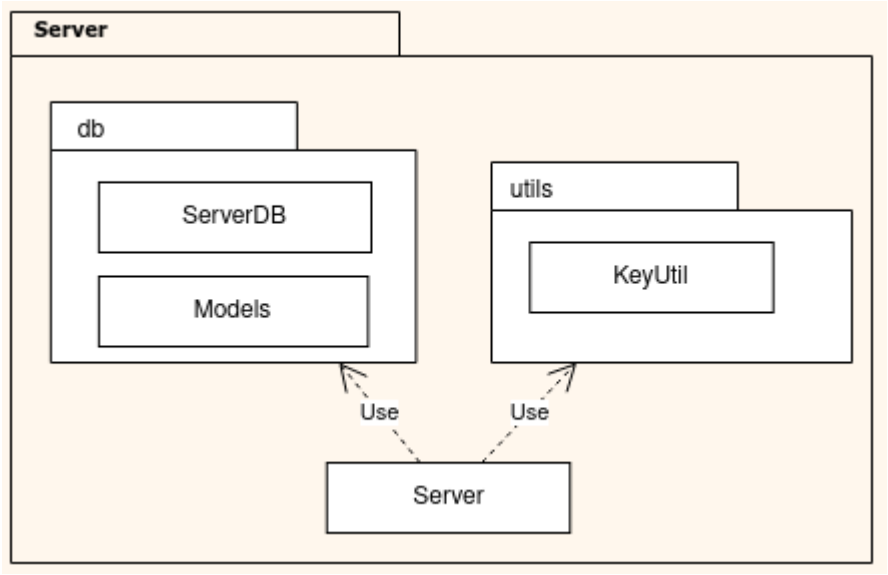

Рисунок 3.5 – Діаграма пакетів сервера

Ця діаграма пакетів сервера має 2 пакети та 1 звичайний компонент. "Server" є простим компонентом, котрий використовує пакети "db" та "utils", що показано відповідними зв’язками, для з’єднання з БД сервера та для забезпечення шифрування даних.

3.2 Діаграми баз даних

Діаграма бази даних є важливим інструментом при розробці програмного забезпечення, особливо коли мова йде про застосунки, які використовують бази даних для збереження і обробки інформації.

Для забезпечення правильної роботи застосунка месенджера з функцією локального збереження історії листування необхідним є створити 3 діаграми для трьох баз даних. Дві бази даних будуть використовуватись в клієнтському ПЗ, а третя – в серверному.

Представлена нижче діаграма (рис. 3.6) відображає структуру клієнтської бази даних – "Launcher".

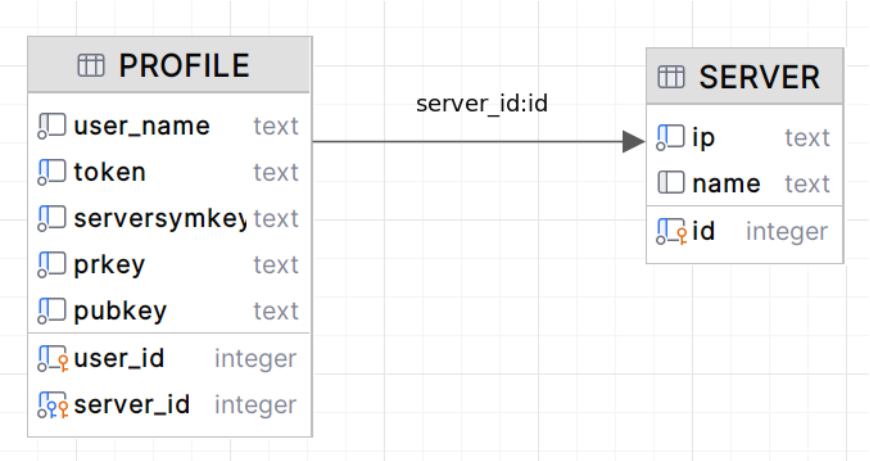

Рисунок 3.6 – Діаграма клієнтської бази даних "Launcher"

Продемонстрована БД необхідна для зберігання інформації про авторизовані профілі всередині лаунчеру, для швидкого доступу до акаунтів клієнта. База даних має дві таблиці, "Profile" для зберігання інформації про профілі та "Server" для зберігання інформації про сервер, який прив’язаний до певного профілю. Саме тому ці таблиці мають між собою зв’язок.

На діаграмі нижче (рис. 3.7) показано структуру клієнтської бази даних – "Client". Ця база даних буде використовуватись для збереження такої інформації: історія листування, список контактів та заблокованих користувачів.

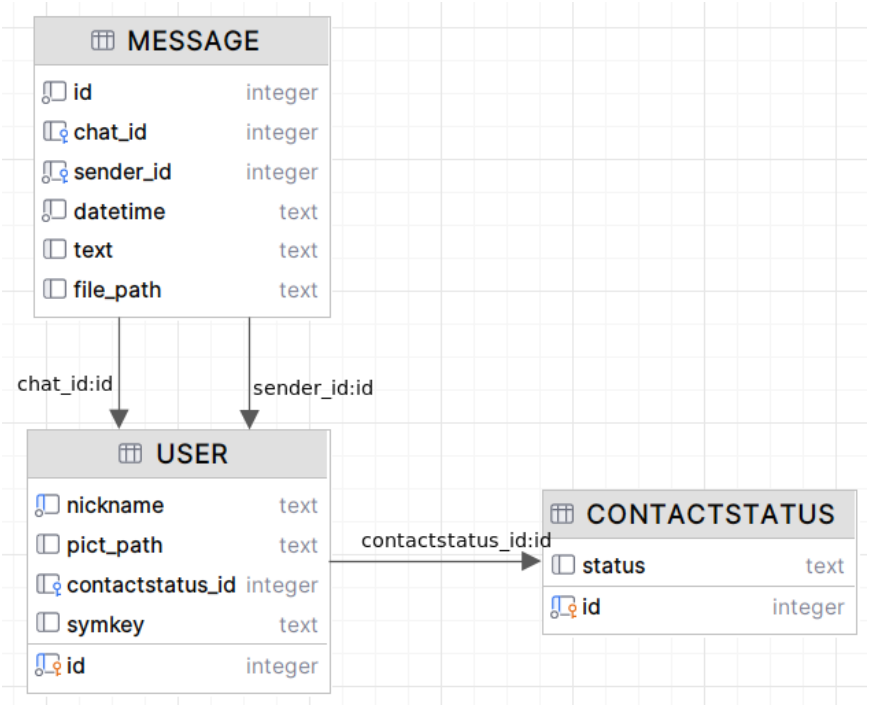

Рисунок 3.7 – Діаграма клієнтської бази даних "Client"

Нижче наведено діаграму для серверної БД – "Server" (рис. 3.8).

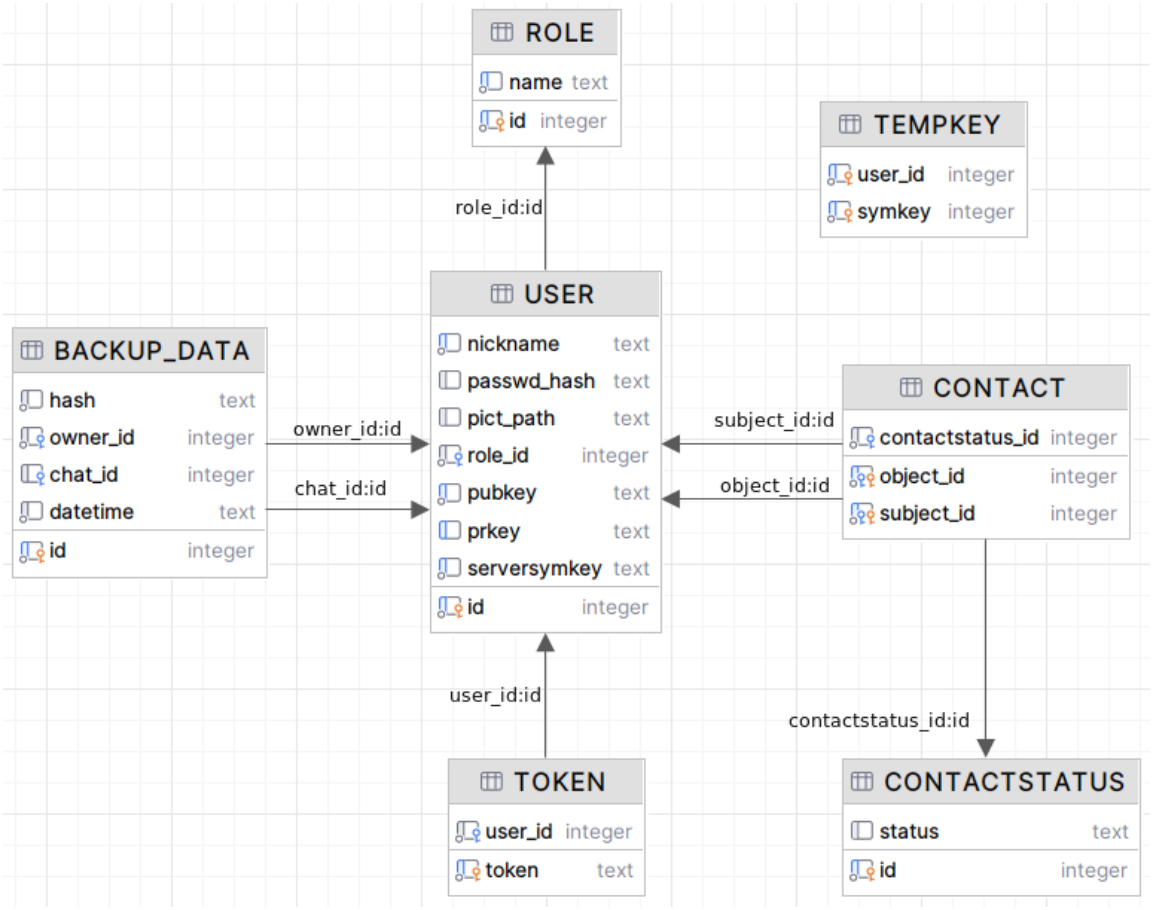

Рисунок 3.8 – Діаграма серверної бази даних "Server"

Структура бази даних "Server" відповідає за збереження такої інформації: список зареєстрованих профілів та їх необхідну інформацію, токенів для підтвердження ідентичності користувачів, список контактів та заблокованих профілів користувача, дані про збереження локальної історії листування для валідації при відновленні, тимчасові ключі шифрування для реєстрації/авторизації.

3.3 Огляд стеку технологій

Задля розробки месенджеру з функцією локального збереження історії існування буде застосовано наступні технології:

- мова програмування: Python;
- GUI: PyQt [11];
- СКБД: SQLite;
- інші бібліотеки: cryptography [12], SQLAlchemy, socket.

Мова програмування

Python було обрано як основну мову програмування для розробки месенджера з кількох вагомих причин. Насамперед, Python відомий своєю простотою та зрозумілим синтаксисом, що робить його ідеальним для швидкої розробки та прототипування. Тому ця мова легко дозволить писати та підтримувати код, знижуючи кількість помилок і скорочуючи час на розробку.

Окрім цього, Python має величезну кількість бібліотек та фреймворків, які спрощують реалізацію багатьох функцій. Наприклад, для шифрування та забезпечення безпеки буде використано бібліотеку cryptography, а для роботи з базою даних – SQLAlchemy. Це дозволяє не витрачати час на написання низькорівневого коду і зосередитися на основних функціональних можливостях месенджера.

Для реалізації функціональності передачі повідомлень по мережі Python надає бібліотеку socket, яка дозволяє легко створювати клієнт-серверні застосунки. Це забезпечить надійну комунікацію між користувачами месенджера. Використання бібліотеки cryptography дозволяє впровадити сучасні алгоритми шифрування для захисту повідомлень та даних користувачів [13; 14].

Ще однією перевагою Python є його кросплатформеність, що дозволяє запускати застосунок на різних операційних системах без значних змін у коді. Це забезпечує широке охоплення користувачів і зручність у розробці.

Підбиваючи підсумки, можна сказати, що вибір мови Python для розробки месенджера з функцією локального збереження історії листування є обґрунтованим і ефективним рішенням, яке забезпечує швидкість розробки, безпеку та зручність використання.

Технологія графічного інтерфейсу користувача

Для створення графічного інтерфейсу користувача було обрано PyQt, оскільки цей фреймворк має низку значних переваг. PyQt є одним з найпопулярніших інструментів для розробки GUI на Python, що базується на бібліотеці Qt, яка відома своєю потужністю та гнучкістю.

По-перше, PyQt, як і сам Python забезпечує кросплатформеність, дозволяючи розробляти застосунки, які будуть працювати на різних операційних системах, таких

як Windows, macOS та Linux.

По-друге, PyQt пропонує широкий набір інструментів та віджетів для створення інтуїтивно зрозумілого інтерфейсу. Це включає все необхідне для розробки функціональних інтерфейсів, від простих кнопок та полів вводу до складних компонентів, таких як таблиці, діаграми та панелі інструментів. Завдяки цьому можна створити зручний та функціональний інтерфейс месенджера, який буде комфортним для користування.

По-третє, PyQt підтримує роботу з багатопотоковістю, що є важливим, якщо треба забезпечити плавну та швидку роботу інтерфейсу користувача. Це дозволяє виконувати фонові операції, такі як обробка повідомлень або взаємодія з базою даних, не блокуючи основний потік застосунку і не перериваючи взаємодію з користувачем.

Крім того, PyQt має гарну документацію та активну спільноту, що значно полегшує процес навчання та розробки. Наявність великої кількості прикладів, навчальних матеріалів та форумів забезпечує швидке вирішення проблем та впровадження нових функцій у застосунок. Це також означає, що в разі виникнення складних завдань або помилок завжди можна знайти підтримку або готові рішення.

Таким чином, вибір PyQt для створення графічного інтерфейсу користувача є обґрунтованим рішенням, яке забезпечує кросплатформеність, багатий набір інструментів, підтримку багатопотоковості, а також доступ до широкої спільноти та ресурсів.

База даних

Для збереження даних месенджера, як для сервера, так і для клієнта, було обрано СКБД SQLite. Це рішення має кілька ключових переваг, які роблять SQLite оптимальним вибором.

Насамперед, SQLite є вбудованою системою керування базами даних, що означає, що вона не потребує налаштування окремого серверного оточення. Це значно спрощує процес встановлення та обслуговування застосунка, оскільки всі необхідні файли бази даних знаходяться в одному місці.

Також ця база даних займає мінімум місця на диску та має невеликі вимоги до

пам'яті, що робить її ідеальною для використання на будь-яких пристроях, включаючи ті, що мають обмежені ресурси.

SQLite забезпечує високу продуктивність і ефективність при роботі з невеликими та середніми обсягами даних. Тому завдяки своїй простоті та оптимізації, SQLite може швидко виконувати запити та операції з даними, що є критичним для застосунків, які потребують швидкого доступу до інформації. Це дозволяє месенджеру миттєво зберігати та завантажувати історію листування без затримок.

Також варто зазначити, що SQLite підтримує всі основні можливості реляційних баз даних, такі як транзакції, забезпечення цілісності даних та робота з складними запитами. Це дозволяє реалізувати складні структури даних та забезпечити надійність збереження інформації. Використання SQLite забезпечує надійність та консистентність даних, що є важливим для збереження історії листування.

Останнім, але не менш важливим фактором є те, що SQLite є відкритим програмним забезпеченням. Це означає, що вона не потребує ліцензійних платежів, що знижує загальні витрати на розробку та підтримку застосунку.

Таким чином, вибір SQLite для збереження даних месенджера є обґрунтованим і ефективним рішенням, яке забезпечує простоту використання, високу продуктивність, надійність, компактність та кросплатформеність.

Висновки до розділу 3

У третьому розділі КРБ було створено UML-діаграми класів, компонентів та пакетів, що дозволило детально спроєктувати структуру та поведінку системи, забезпечуючи її модульність, масштабованість та легкість у підтримці.

Вибір технологій, таких як Python, SQLite, PyQt та інших бібліотек, забезпечує гнучкість, продуктивність та зручність розробки месенджера з функцією локального збереження історії листування.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

4.1 Програмна реалізація основних частин системи сервера

Необхідним завданням для реалізації коду сервера є створення та налаштування сокетів для прийому з'єднань, обробки вхідних запитів від клієнтів та забезпечення безпечного обміну даними між клієнтами і сервером, що продемонстровано нижче (рис. 4.1).

```
def start(self):
    self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.server_socket.bind((self.host, self.port))
    self.server_socket.listen()

    print(f"Server is listening on {self.host}:{self.port}")

    while True:
        ready_to_read, _, _ = select.select([self.server_socket] + self.client_sockets, [], [])

        for sock in ready_to_read:
            if sock == self.server_socket:
                client_socket, client_address = self.server_socket.accept()
                self.client_sockets.append(client_socket)
                print(f"Client connected from {client_address}")
            else:
                try:
                    data = sock.recv(8192)
                    if data:
                        request = pickle.loads(data)
                        response = self.handle_request(request)
                        if response:
                            sock.sendall(pickle.dumps(response))
                    else:
                        if sock in self.client_sockets:
                            self.client_sockets.remove(sock)
                        print("Connection closed by client")
                except Exception as e:
                    print(f"Error reading from client: {e}")
                    continue
```

Рисунок 4.1 – Функція запуску сервера

При запуску, за який відповідає функція "server", сервер створює сокет, прив'язує його до вказаної адреси та починає прослуховувати вхідні з'єднання. Головний цикл сервера використовує select.select для моніторингу готових до читання сокетів, включаючи основний серверний сокет і всі активні клієнтські сокети. Коли новий клієнт підключається, його сокет приймається, додається до списку клієнтів і відображається інформація про підключення. Кожен клієнтський сокет, який готовий до читання, приймає дані від клієнта, обробляє їх запит через виклик handle_request, та відправляє відповідь клієнту. Якщо з'єднання з клієнтом закривається або виникає помилка під час читання, сокет видаляється зі списку активних клієнтів, а інформація про помилку виводиться у консоль.

Наступна частина коду (рис. 4.2) включає в себе механізм обробки запитів від клієнтів через визначені дії "actions" та функцію "handle_request". Це дозволяє серверу взаємодіяти з клієнтами, реагуючи на різноманітні запити, такі як авторизація користувача, обмін повідомленнями та файлами, керування контактами та інші операції.

```
class Server:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.server_socket = None
        self.client_sockets = []
        self.messages = {}
        self.server_db = ServerDB()

        self.actions = {
            "login_exchange_tempkeys": self.process_login_exchange_tempkeys,
            "login_verify_passwd": self.process_login_verify_passwd,
            "verify_user": self.process_verify_user,
            "signup_new_symkey": self.process_signup_new_symkey,
            "signup_finish": self.process_signup_finish,
            "search_user": self.process_search_user,
            "manage_contacts": self.process_manage_contacts,
            "send_message": self.process_send_message,
            "send_file": self.process_send_file,
            "download_file": self.process_download_file,
            "create_backup": self.process_create_backup,
            "restore_backup": self.process_restore_backup,
            "edit_profile": self.process_edit_profile,
        }

        self.db = ServerDB("db/other/server.sqlite")
        self.keyutil = KeyUtil()

    def start(self):...

    def handle_request(self, request):
        action = request.get("action")
        if action in self.actions:
            return self.actions[action](request)
        else:
            return {"error": "Unknown action"}
```

Рисунок 4.2 – Код обробки запитів на сервер

Для користуванням повним функціоналом застосунку в першу чергу треба пройти етап реєстрацій. Код нижче демонструє дві функції, які відповідають за обробку запитів "signup_new_symkey" (рис. 4.3) та "signup_finish" (рис. 4.4) відповідно.

При отриманні запиту "signup_new_symkey" на реєстрацію нового користувача, відбуваються такі дії:

– перевіряються дані на їх наявність, а саме ім'я користувача та публічний ключ;

– перевіряється унікальність імені користувача в базі даних;

– генерується новий симетричний ключ для користувача;

– створюється новий запис користувача у базі даних з використанням отриманого імені, публічного ключа та симетричного ключа;

– симетричний ключ шифрується асиметрично з використанням публічного ключа користувача;

– ідентифікатор нового користувача також шифрується асиметрично та повертається разом з зашифрованим симетричним ключем у відповідь.

```
def process_signup_new_symkey(self, received_data):
    nickname = received_data.get("nickname")
    pubkey = received_data.get("pubkey")

    if not nickname or not pubkey:
        return {"error": "Invalid signup_new_symkey request. Expected format: signup_new_symkey <nickname> <pubkey>"}

    existing_user = self.db.get_user_by_name(nickname)

    if existing_user:
        return {"error": f"Nickname '{nickname}' is already taken."}

    new_symkey = self.keyutil.gen_symkey()
    new_user = self.db.create_user(nickname, pubkey, new_symkey)

    if not new_user:
        return {"error": "Failed to create new user."}

    encrypted_symkey = self.keyutil.encrypt_data_asym(new_symkey, pubkey)
    encrypted_user_id = self.keyutil.encrypt_data_asym(str(new_user.get("id")), pubkey)
    return {"message": "New user and symmetric key created successfully.", "data": {"symkey": encrypted_symkey, "user_id": encrypted_user_id}}
```

Рисунок 4.3 – Код обробки запиту "signup_new_symkey"

При отриманні правильних даних від сервера, клієнт виконає запит "signup_finish", для завершення реєстрації, де відбуватимуться такі дії:

– перевіряється коректність отриманих даних (ідентифікатор користувача, зашифрований хеш паролю та зашифрований приватний ключ);

– отримується запис користувача за його ідентифікатором з бази даних;

– розшифровується симетричний ключ користувача, який використовується для дешифрації хеша паролю та приватного ключа;

– проводиться перевірка відповідності отриманого хеша паролю хешу паролю користувача в базі даних;

– якщо дані вірні, генерується новий токен для користувача, який шифрується з використанням симетричного ключа користувача.

– токен повертається як частина відповіді сервера.

```python
def process_signup_finish(self, received_data):
    user_id = received_data.get("user_id")
    passwd_hash = received_data.get("passwd_hash")
    prkey = received_data.get("prkey")

    if not user_id or not passwd_hash or not prkey:
        return {"error": "Invalid signup_finish request. Expected format: signup_finish <user_id> <passwd_hash> <prkey>"}

    user = self.db.get_user(user_id)

    if not user:
        return {"error": "User not found."}

    symkey = user.get("serversymkey")
    decrypted_passwd_hash = self.keyutil.decrypt_data_sym(passwd_hash, symkey)
    decrypted_prkey = self.keyutil.decrypt_data_sym(prkey, symkey)

    if not decrypted_passwd_hash or not decrypted_prkey:
        return {"error": "Failed to decrypt symmetric key or private key."}

    if not self.db.complete_create_user(user_id, decrypted_passwd_hash, decrypted_prkey):
        return {"error": "Failed to register user."}

    token = self.keyutil.gen_token()
    self.db.create_token(user_id, token)
    encrypted_token = self.keyutil.encrypt_data_sym(token, symkey)
    return {"message": "User registered successfully.", "data": {"token": encrypted_token}}
```

Рисунок 4.4 – Код обробки запиту "signup_finish"

У випадку, якщо користувач вже був зареєстрований в системі, йому треба лише пройти авторизацію, яка здійснюється послідовним викликом таких запитів на сервер: "login_exchange_tempkeys", "login_verify_passwd", "verify_user".

Алгоритм обробки запиту "login_exchange_tempkeys" на сервері:

– перевіряється коректність отриманих даних (ім'я користувача та тимчасовий публічний ключ;

– за нікнеймом з бази даних отримується об’єкт користувача;

– генерується тимчасовий симетричний ключ для користувача та зберігається в базі даних;

– тимчасовий симетричний ключ шифрується асиметрично з використанням тимчасового публічного ключа;

– повертається відповідь з успішно згенерованим тимчасовим симетричним ключем та ідентифікатором користувача.

Код цього алгоритму показано нижче (рис. 4.5).

```python
def process_login_exchange_tempkeys(self, received_data):
    user_name = received_data.get("user_name")
    temp_pubkey = received_data.get("temp_pubkey")
    if not user_name or not temp_pubkey:
        return {"error": "Invalid login_exchange_tempkeys. Expected format: login_exchange_tempkeys <user_name> <temp_pubkey>"}

    user = self.db.get_user_by_name(user_name)

    if not user:
        return {"error": "User not found."}

    temp_symkey = self.keyutil.gen_symkey()
    self.db.create_tempkey(user.get("id"), temp_symkey)
    encrypted_symkey = self.keyutil.encrypt_data_asym(temp_symkey, temp_pubkey)
    return {"message": "Temp symkey generated.", "data": {"user_id": user.get("id"), "encrypted_symkey": encrypted_symkey}}
```

Рисунок 4.5 – Код обробки запиту "login_exchange_tempkeys"

Після того як клієнт отримає відповідь без помилки, наступним буде виконуватись запит "login_verify_passwd", який включає в себе такі дії:

– перевіряється коректність отриманих даних (ідентифікатор користувача та зашифрований хеш паролю);

– отримується запис користувача з бази даних за його ідентифікатором;

– отримується тимчасовий симетричний ключ користувача з бази даних, створений в попередньому запиті;

– розшифровується отриманий зашифрований хеш паролю за допомогою тимчасового симетричного ключа;

– порівнюється розшифрований хеш паролю з хешем паролю користувача в базі даних;

– якщо пароль вірний, генерується новий токен для користувача та зберігається в базі даних разом із симетричним ключем сервера, приватним ключем та публічним ключем користувача;

– всі дані шифруються симетрично тимчасовим симетричним ключем;

– тимчасовий симетричний ключ видаляється з бази даних;

– повертається відповідь про успішний вхід користувача з авторизованим токеном та іншими даними.

Код запиту "login_verify_passwd" наведено нижче (рис. 4.6)

```
def process_login_exchange_tempkeys(self, received_data):
    user_name = received_data.get("user_name")
    temp_pubkey = received_data.get("temp_pubkey")
    if not user_name or not temp_pubkey:
        return {"error": "Invalid login_exchange_tempkeys. Expected format: login_exchange_tempkeys <user_name> <temp_pubkey>"}

    user = self.db.get_user_by_name(user_name)

    if not user:
        return {"error": "User not found."}

    temp_symkey = self.keyutil.gen_symkey()
    self.db.create_tempkey(user.get("id"), temp_symkey)
    encrypted_symkey = self.keyutil.encrypt_data_asym(temp_symkey, temp_pubkey)
    return {"message": "Temp symkey generated.", "data": {"user_id": user.get("id"), "encrypted_symkey": encrypted_symkey}}
```

Рисунок 4.6 – Код обробки запиту "login_verify_passwd"

Останнім етапом авторизації є виконання запиту "verify_user" з наступним алгоритмом:

– перевіряється коректність отриманих даних (ідентифікатор користувача, зашифрований токен та зашифрований публічний ключ);

– отримується запис користувача з бази даних за його ідентифікатором;

– отримується симетричний ключ користувача з бази даних;

– розшифровуються отримані зашифрований токен та публічний ключ за допомогою симетричного ключа;

– перевіряється валідність розшифрованого токена та публічного ключа у базі даних;

– повертається відповідь про успішність або помилку підтвердження користувача.

Код описаного алгоритму продемонстровано нижче (рис. 4.7)

```
def process_verify_user(self, received_data):
    user_id = received_data.get("user_id")
    encrypted_token = received_data.get("encrypted_token")
    encrypted_pubkey = received_data.get("encrypted_pubkey")

    if not user_id or not encrypted_token or not encrypted_pubkey:
        return {"error": "Invalid verify_user request. Expected format: verify_user <user_id> <encrypted_token> <encrypted_pubkey>"}

    user = self.db.get_user(user_id)

    if not user:
        return {"error": "User not found."}

    symkey = user.get("serversymkey")

    if not symkey:
        return {"error": "Failed to get symkey for user."}

    decrypted_token = self.keyutil.decrypt_data_sym(encrypted_token, symkey)
    decrypted_pubkey = self.keyutil.decrypt_data_sym(encrypted_pubkey, symkey)

    if self.db.verify_token(user_id, decrypted_token) and self.db.verify_pubkey(user_id, decrypted_pubkey):
        return {"message": "User verification successful.", "data": {"success": True}}
    else:
        return {"error": "Invalid token or pubkey."}
```

Рисунок 4.7 – Код обробки запиту "verify_user"

Кожна з інших функцій, які не було детально описано, реалізують конкретну логіку обробки даних, яка відповідає за виконання різних операцій на сервері, таких як обмін повідомленнями та файлами, створення та відновлення резервних копій, керування контактами, списком заблокованих користувачів, а також редагування профілю користувача, які взаємодіють з серверною БД. Кожна така функція повертає відповідь на клієнтський запит у форматі JSON із зазначенням статусу операції та відповідних даних, якщо такі є, або повідомлення про помилку у випадку невдалого виконання операції.

4.2 Програмна реалізація основних частин системи клієнта

Реалізація класу "Client"

Для взаємодії з сервером клієнтська частина застосунку повинна мати відповідні механізми з'єднання, відправки запитів і отримання відповідей. Ці функції реалізовані за допомогою класу "Client" (рис. 4.8). Перш за все, конструктор класу ініціалізує змінні для збереження адреси сервера та порту, після чого відбувається спроба підключитися до сервера через метод "connect()".

При успішному підключенні до серверу, виводиться повідомлення про підключення до сервера. Якщо підключення вже існує або відбувається помилка підключення, виводиться відповідне повідомлення.

Метод "send_request" відповідає за відправку серіалізованого запиту до сервера через створене з'єднання. Він також очікує отримати відповідь від сервера, яку потім розшифровує з формату "pickle" за допомогою методу "receive_data".

Метод "check_for_updates" є особливістю цього класу, який періодично відправляє запити на сервер для перевірки наявності оновлень. Цикл "while True" забезпечує постійну перевірку, інтервал між запитами становить 5 секунд.

Крім цього, клас має методи для зміни сервера "change_server", перевірки з'єднання "check_connection", закриття з'єднання "close", а також для оновлення даних з'єднання "set_connection_data".

```
class Client:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.client_socket = None
        self.connect()

    def set_connection_data(self, host, port):
        if self.host == host and self.port == port:
            return False
        self.host = host
        self.port = port
        return True

    def check_connection(self):
        return self.client_socket is not None

    def connect(self):
        if not self.check_connection():
            self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            try:
                self.client_socket.connect((self.host, self.port))
                print(f"Connected to server at {self.host}:{self.port}")
                return True
            except Exception as e:
                print(f"Failed to connect to server: {e}")
                return False
        print("Failed to connect. Connection exist")
        return False

    def send_request(self, request_data):
        try:
            serialized_data = pickle.dumps(request_data)
            self.client_socket.sendall(serialized_data)
            received_data = self.receive_data()
            response, data = pickle.loads(received_data)
            return response, data
        except Exception as e:
            print(f"An error occurred: {e}")
            return None, None

    def receive_data(self):
        received_data = self.client_socket.recv(8192)
        return received_data

    def close(self):
        if self.check_connection():
            self.client_socket.close()
            self.client_socket = None
            print("Disconnected from server")
            return True
        print("Failed to close connection. No connection to close")
        return False

    def change_server(self, host, port):
        if self.set_connection_data(host, port):
            self.close()
            self.connect()
        return self.check_connection()
```

Рисунок 4.8 – Код класу "Client"

Реалізація класу "Context"

Клієнтська частина застосунку має головний клас – "Context", який відповідає за керування вікнами PyQt [8], зв'язок з сервером через клас "Client", та доступ до локальної бази даних через "LauncherDB". У конструкторі "Context" ініціалізуються основні компоненти: клієнт для зв'язку з сервером, об'єкт для роботи з базою даних, а також налаштовується інтерфейс за допомогою PyQt. Завдяки методам "open_messenger" і "exit_messenger" можливий перехід між вікнами лаунчера та месенджера, які представлені відповідними класами "LauncherWindow" та "MessengerWindow". Після ініціалізації у блоку "main" створюється екземпляр "Context", який запускає головний цикл застосунку PyQt, що керує відображенням і

взаємодією з користувачем до моменту закриття програми. Код цього класу наведено нижче (рис. 4.9).

```
class Context:
    def __init__(self):
        self.client = Client("localhost", 12345)
        self.ldb = LauncherDB('db/other/launcher.sqlite')

        self.app = QApplication(sys.argv)
        self.load_styles()

        self.launcher_window = LauncherWindow(self)
        self.messenger_window = None
        self.launcher_window.show()

    def open_messenger(self, profile_info, ip):
        self.launcher_window.hide()
        self.launcher_window.clear_all_forms()

        self.messenger_window = MessengerWindow(self, profile_info, ip)
        self.messenger_window.show()
        print("Opened Messenger")

    def exit_messenger(self):
        self.messenger_window.hide()
        self.messenger_window = None
        print("Exited Messenger")
        self.launcher_window.show()

    def load_styles(self):
        with open("res/styles/main.qss", "r") as f:
            self.app.setStyleSheet(f.read())

    def run(self):
        sys.exit(self.app.exec())

if __name__ == "__main__":
    context = Context()
    context.run()
```

Рисунок 4.9 – Код класу "Context"

Клієнтська частина застосунку поділяється на два вікна: "LauncherWindow" та "MessengerWindow". Кожне з них має власний набір представлень та контролерів, а також обидва наслідують клас "BaseWindow", яке відповідає за відцентрування вікон та задавання заголовка.

Реалізація класу "LauncherWindow"

Клас LauncherWindow (рис. 4.10) має декілька основних функцій, які визначають його поведінку і взаємодію з іншими частинами програми.

Конструктор класу – ініціалізує всі необхідні компоненти вікна, такі як віджети ("LauncherMenuView", "LoginView", "SignupView") і контролери ("LoginController", "SignupController"). Також встановлюється початковий вигляд вікна і показується "LauncherMenuView".

Для налаштування графічного інтерфейсу вікна лаунчера використовується метод "setup_ui". Він створює "QStackedWidget" і додає до нього всі необхідні віджети, так щоб їх можна було переключати. Також встановлюється початковий розмір вікна і поточний віджет.

Метод "setup_connections", який разом з "setup_ui", буде неодноразово зустрічатись в коді клієнту, встановлює зв'язки між сигналами і відповідними слотами, які реагують на ці сигнали. Використовуються lambda-функції для передачі параметрів до методу switch_view, який відповідає за зміну активного віджету у "QStackedWidget".

Щоб забезпечити функціонал зміни активного віджета у "QStackedWidget" на той, який передається як параметр "view", створено функцію "switch_view". В залежності від вибраного віджету відбувається налаштування розмірів вікна і його центрування. Також викликаються методи контролерів для завантаження даних або очищення форми при необхідності.

```
class LauncherWindow(BaseWindow):
    def __init__(self, context):
        super().__init__(context, "Launcher")

        self.launcher_menu_view = LauncherMenuView()
        self.login_view = LoginView()
        self.signup_view = SignupView()

        self.login_controller = LoginController(self.login_view, context)
        self.signup_controller = SignupController(self.signup_view, context)

        self.stacked_widget = QStackedWidget()

        self.setup_ui()

        self.setup_connections()

    def setup_ui(self):
        self.stacked_widget.addWidget(self.launcher_menu_view)
        self.stacked_widget.addWidget(self.login_view)
        self.stacked_widget.addWidget(self.signup_view)

        self.setCentralWidget(self.stacked_widget)
        self.setFixedSize(200, 120)
        self.stacked_widget.setCurrentWidget(self.launcher_menu_view)

    def setup_connections(self):
        self.launcher_menu_view.signup_button.clicked.connect(lambda: self.switch_view(self.signup_view))
        self.launcher_menu_view.login_button.clicked.connect(lambda: self.switch_view(self.login_view))

        self.login_view.back_button.clicked.connect(lambda: self.switch_view(self.launcher_menu_view))
        self.signup_view.back_button.clicked.connect(lambda: self.switch_view(self.launcher_menu_view))

    def switch_view(self, view):
        previous_center = self.frameGeometry().center()
        self.stacked_widget.setCurrentWidget(view)
        if view == self.login_view:
            self.login_controller.load_profiles()
            self.setFixedSize(530, 600)
            self.move_prevcenter(previous_center)
        elif view == self.signup_view:
            self.setFixedSize(350, 330)
            self.move_prevcenter(previous_center)
        else:
            self.setFixedSize(200, 120)
            self.center_window()

        self.clear_all_forms()

    def clear_all_forms(self):
        self.login_controller.clear_form()
        self.signup_controller.clear_form()

    def show(self):
        super().show()
        self.switch_view(self.launcher_menu_view)
```

Рисунок 4.10 – Код класу "LauncherWindow"

"LauncherMenuView" має в собі всього лише 2 кнопки, для вибору реєстрації чи авторизації, також ці представлення та контролери мають схожий механізм, тому буде більш детально описано представлення "LoginView" і контролер "LoginController".

Реалізація класу "LoginView"

У конструкторі класу "LoginView" налаштовуються основні віджети і контейнери для відображення елементів інтерфейсу. Зокрема, створюються і налаштовуються таблиця "QTableWidget" для відображення профілів вже авторизованих користувачів, текстові поля для введення інформації (сервера, імені сервера, нікнейму, паролю), а також кнопки для відправки форми та повернення назад.

Для заповнення таблиці даними авторизованих профілів користувачів, використовується метод "populate_profiles". Він приймає список профілів, який надаватиме контролер, і додає їх до таблиці у відповідних рядках і стовпцях. Для кожного профілю створюються віджети "QTableWidgetItem" для відображення адреси та імені сервера, ім'я користувача, а також кнопка "Remove" для видалення профілю, як авторизованого, з лаунчера.

Якщо казати про UI та взаємодію користувача з ним, цей процес відбувається через обробники подій. Наприклад, подвійне натискання на елемент таблиці викликає метод "open_user", який емітує сигнал "open_user_signal" з адресою серверу і ім'ям користувача. Кнопка "Submit" викликає метод "login_submit", який емітує сигнал "login_submit_signal" із введеними даними для входу. Кнопка "Remove" в таблиці викликає метод "remove_profile", який емітує сигнал "remove_profile_signal" для видалення відповідного профілю зі списку авторизованих користувачів.

Реалізація класу "LoginView" наведена нижче (рис. 4.11), проте відображаючи тільки основну логіку та функції, через тривіальність коду з налаштуванням елементів графічного інтерфейсу та стилів.

```python
class LoginView(QWidget):
    open_user_signal = pyqtSignal(str, str)
    remove_profile_signal = pyqtSignal(str, str)
    login_submit_signal = pyqtSignal(str, str, str, str)

    def __init__(self):...
    def setup_ui_table(self):...
    def setup_ui_form(self):...
    def setup_ui_form_layout(self):...
    def setup_ui_layout(self):...
    def setup_styles(self):...
    def setup_connections(self):
        self.profile_table.itemDoubleClicked.connect(self.open_user)
        self.submit_button.clicked.connect(self.login_submit)

    def populate_profiles(self, profiles):
        self.profile_table.setRowCount(len(profiles))
        for row, profile in enumerate(profiles):
            ip = profile.get('ip', '')
            server_name = profile.get('server_name', '')
            user_name = profile.get('user_name', '')
            ip_item = QTableWidgetItem(ip)
            ip_item.setFlags(ip_item.flags() & ~Qt.ItemFlag.ItemIsEditable)

            server_name_item = QTableWidgetItem(server_name)
            server_name_item.setFlags(server_name_item.flags() & ~Qt.ItemFlag.ItemIsEditable)

            user_name_item = QTableWidgetItem(user_name)
            user_name_item.setFlags(user_name_item.flags() & ~Qt.ItemFlag.ItemIsEditable)
            remove_button = QPushButton("Remove")
            remove_button.clicked.connect(lambda _, ip=ip, user_name=user_name: self.remove_profile(ip, user_name))
            self.profile_table.setCellWidget(row, 3, remove_button)

            self.profile_table.setItem(row, 0, ip_item)
            self.profile_table.setItem(row, 1, server_name_item)
            self.profile_table.setItem(row, 2, user_name_item)
        for i in range(3):
            self.profile_table.setColumnWidth(i, 124)

        #self.profile_table.resizeColumnsToContents()

    def open_user(self, item):
        row = item.row()
        ip = self.profile_table.item(row, 0).text()
        user_name = self.profile_table.item(row, 2).text()
        self.open_user_signal.emit(ip, user_name)

    def login_submit(self):
        nickname = self.nickname_input.text()
        password = self.password_input.text()
        server_ip = self.server_input.text()
        server_name = self.servername_input.text()
        self.login_submit_signal.emit(nickname, password, server_ip, server_name)

    def remove_profile(self, ip, user_name):
        self.remove_profile_signal.emit(ip, user_name)
```

Рисунок 4.11 – Код класу "LoginView"

Реалізація класу "LoginController"

Клас "LoginController", який є підкласом "BaseLauncherController", відповідає за обробку логіки взаємодії між моделлю даних ("LauncherDB"), користувацьким інтерфейсом ("LoginView") і зовнішнім сервером. Основна логіка цього класу наступна:

У конструкторі цього ініціалізуються змінні для профілів користувачів та налаштовуються з'єднання з сигналами ("open_user_signal", "login_submit_signal", "remove_profile_signal") з відповідними обробниками подій (рис. 4.12):

– "handle_open_user" виконує перевірку та з'єднання з клієнтом за адресою сервера й відкриває профіль користувача та передає інформацію в контекст програми для відкриття месенджера;

– "handle_login_submit" перевіряє введені дані, забезпечує з'єднання з

сервером і виконує процедуру входу користувача і при успішному вході відкриває месенджер;

– "handle_remove_profile" видаляє вибраний профіль з бази даних лаунчера і після видалення оновлює список профілів.

```
def setup_connections(self):
    self.view.open_user_signal.connect(self.handle_open_user)
    self.view.login_submit_signal.connect(self.handle_login_submit)
    self.view.remove_profile_signal.connect(self.handle_remove_profile)

def handle_open_user(self, ip, user_name):
    if not self.ensure_client_connection(ip):
        print("Ensure client connection failed")
        return
    current_profile_info = self.open_user(user_name, ip)
    if current_profile_info:
        print("Open logged in user successful")
        self.context.open_messenger(current_profile_info, ip)
    else:
        print("Open logged in user failed")

def handle_login_submit(self, nickname, password, server_ip, server_name):
    if not self.validate_input(nickname, password, server_ip):
        return

    if not self.ensure_client_connection(server_ip):
        print("Ensure client connection failed")
        return

    current_profile_info = self.login(nickname, password, server_ip, server_name)
    if current_profile_info:
        print("Login successful")
        self.context.open_messenger(current_profile_info, server_ip)
    else:
        print("Login failed")

def handle_remove_profile(self, ip, user_name):
    if self.ldb.del_profile(ip, user_name):
        print("Profile deleted from launcher database.")
    else:
        print("Failed to delete profile.")
    self.load_profiles()
```

Рисунок 4.12 – Код обробників подій класу "LoginController"

В цьому контролері також існують ще такі методи: методи роботи з профілями, методи авторизації та валідації.

Методи роботи з профілями такі:

– "load_profiles" завантажує коротку інформацію про авторизовані профілі з бази даних лаунчера і відображає їх у вигляді таблиці;

– "clear_form" очищує форму введення даних для авторизації.

Клас має такі методи авторизації та валідації:

– "open_user" відкриває профіль користувача за ім'ям та адресою сервера;

– "verify_user" перевіряє користувача, розшифровуючи отримані дані, такі як ідентифікатор користувача, його токен та публічний ключ;

– "login" виконує процедуру входу, генеруючи ключі та обмінюючись інформацією з сервером.

Для прикладу продемонстровано код функції "verify_user" (рис. 4.13), який надсилає запит на сервер та оброблює його відповідь через клас "Client".

```
def verify_user(self, profile, ip):
    user_id = profile.get("user_id")
    pubkey = profile.get("pubkey")
    symkey = profile.get("serversymkey")
    token = profile.get("token")
    user_name = profile.get("user_name")

    encrypted_token = self.keyutil.encrypt_data_sym(token, symkey)
    encrypted_pubkey = self.keyutil.encrypt_data_sym(pubkey, symkey)

    request_data = {
        "action": "verify_user",
        "user_id": user_id,
        "encrypted_token": encrypted_token,
        "encrypted_pubkey": encrypted_pubkey
    }

    response, data = self.client.send_request(request_data)

    if not self.process_response(response, data) or "success" not in data:
        print("Error: Invalid response from the server. Verification failed.")
        if not self.ldb.del_profile(ip, user_name):
            print("Failed to delete profile.")
        else:
            self.load_profiles()
            print("Profile deleted from launcher database.")
        return None

    return profile
```

Рисунок 4.13 – Код класу "LoginController"

Реалізація класу "MessengerWindow"

Клас за своєю структурою не сильно відрізняється від "LauncherWindow", тому буде описано тільки загальну роботу цього класу.

"MessengerWindow" має за мету надати користувачеві зручний інтерфейс для взаємодії з повідомленнями, контактами та налаштуваннями. При запуску та відображенні вікна месенджера створюється основне вікно "QTabWidget", яке містить три вкладки: "Messages", "Contacts" і "Settings". Цей вид віджету дозволяє легко створити декілька сторінок, з вбудованим наданням інтерфейсу для переходу між цими сторінками, у вигляді кнопок.

Вкладки "Messages" та "Contacts" матимуть в собі не просте представлення, як в "Contacts", а спеціальний віджет "QStackedWidget", який дозволяє вміщувати декілька різних віджетів в одній області відображення. Це дозволяє динамічно змінювати контент відповідно до обраної вкладки, відобразивши на поточній вкладці або чаті повідомлення користувачів. Але відрізняється від "QTabWidget" тим, що

"QStackedWidget" дозволяє відображати лише один віджет одночасно, тоді як "QTabWidget" дозволяє користувачам бачити всі вкладки одночасно та обирати потрібну, щоб переглядати її вміст.

Вкладка "Messages" відповідає за відображення чатів, повідомлень користувачів, сторінки користувача, і тому містить всередині "QStackedWidget" ці 3 віджети. "Contacts" відображає список контактів, а також має сторінку відображення профілю користувача з списку контактів. А "Settings" містить інтерфейс для зміни налаштувань профілю.

Для зв'язку між віджетами і головним вікном, так само використовуються сигнали та слоти, як з "LauncherWindow". Наприклад, подвійне клацання на елемент чату чи контакту генерує сигнал, який окрім обробки контролером за необхідністю, передається до MessengerWindow, щоб відобразити відповідну сторінку в "QStackedWidget".

Результати роботи застосунку

Нижче наведені скріншоти роботи вікна лаунчера месенджеру (рис. 4.14).

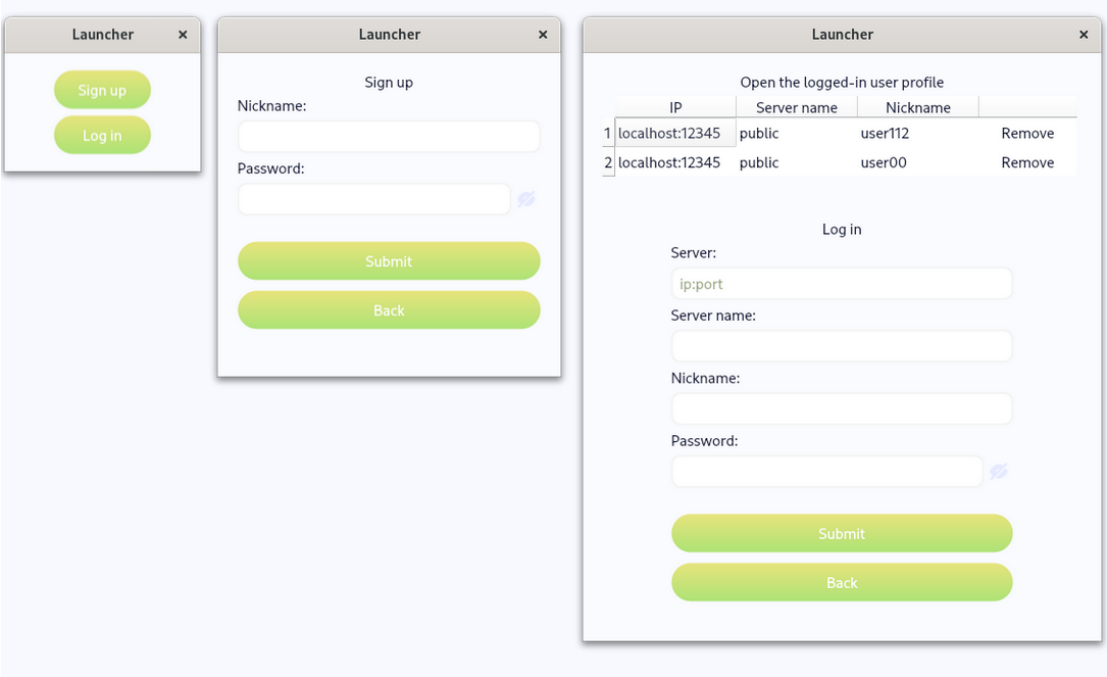

Рисунок 4.14 – Результати роботи вікна лаунчера

Нижче подано резульати роботи вікна лаунчера месенджеру (рис. 4.15), у вигляді скріншотів.

Рисунок 4.15 – Результати роботи вікна месенджера

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи бакалавра розроблено програмну реалізацію основних компонентів системи, включаючи серверну та клієнтську частини. Серверна частина було реалізовано через створення класу, який обробляє запити від клієнтів та забезпечує мережеву взаємодію.

Створено клієнтську частину з включенням класів для здійснення з'єднання з сервером, управління станом програми та взаємодії між різними модулями. Зокрема, створено клас "Client", який відповідає за передачу даних, та клас "Context", котрий відповідає за координацію роботи всіх компонентів.

Крім того, реалізовано такі основні вікна для графічного інтерфейсу: "LauncherWindow" для вікна лаунчера, "MessengerWindow" для основного вікна чату. Для кожного з цих вікон було створено необхідні представлення та контролери.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було розроблено застосунок месенджеру з функцією локального збереження історії листування для підвищення безпеки та зручності комунікації між людьми.

Мета роботи досягнута. Поставлені завдання виконані, а саме:

- досліджено предметну область та проведено аналіз наявних аналогів;
- сформовано специфікації вимог до програмного забезпечення;
- виконано моделювання та проєктування програмного забезпечення;
- розроблено програмну частини коду сервера;
- розроблено клієнтську програмну частини коду.

У ході виконання кваліфікаційної роботи було досліджено предметну галузь, що підтвердило актуальність створення месенджера з функцією локального збереження історії листування.

Також було проведено аналіз наявних месенджерів, їх функціональних можливостей та архітектурних рішень. Це дозволило визначити ключові вимоги до месенджера, що розробляється, з акцентом на локальне збереження історії листування для підвищення безпеки користувачів.

Було сформовано специфікації вимог до програмного забезпечення, що включають в себе детальний опис інтерфейсу користувача та сценарії його використання. Моделювання застосунку з використанням UML-діаграм надало чітке уявлення про структуру та взаємодію компонентів системи.

Проєктування застосунку було здійснено з урахуванням обраного стеку технологій, який включає Python, PyQt, та SQLite, що забезпечує гнучкість та масштабованість продукту. Розробка клієнтської та серверної частин програмного забезпечення відбулася згідно з встановленими вимогами та технічним завданням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1) WhatsApp. URL: https://www.whatsapp.com/ (Last accessed: 08.04.2024).

2) Signal. URL: https://signal.org/ (Last accessed: 08.04.2024).

3) Telegram. URL: https://telegram.org/ (Last accessed: 08.04.2024).

4) PyQt6 Reference Guide. URL: https://www.riverbankcomputing.com/static/Docs/PyQt6/ (Last accessed: 11.04.2024).

5) SQLite Documentation. URL: https://www.sqlite.org/docs.html (Last accessed: 12.04.2024).

6) SQLAlchemy Documentation. URL: https://docs.sqlalchemy.org (Last accessed: 12.04.2024).

7) Python documentation. URL: https://docs.python.org (Last accessed: 09.04.2024).

8) Qt for Python. Qt Documentation. URL: https://doc.qt.io/qtforpython-6/ (Last accessed: 10.04.2024).

9) Wesley E. Transmission Control Protocol (TCP). Request for Comments. № 9293. RFC Editor, 2022. 98 p. URL: https://www.rfc-editor.org/info/rfc9293 (Last accessed: 15.04.2024).

10) Unified Modeling Language Specification Version 2.5.1 / Bock C., others. Object Management Group, 2017. URL: https://www.omg.org/spec/UML/2.5.1/ (Last accessed: 18.05.2024).

11) Willman J. M. Beginning PyQt: A Hands-on Approach to GUI Programming with PyQt6. Apress, 2022. 543 p.

12) Cryptography library documentation. URL: https://cryptography.io (Last accessed: 11.04.2024).

13) PKCS #1: RSA Cryptography Specifications Version 2.2. Request for Comments. № 8017 / Moriarty K., others. RFC Editor, 2016. 78 p. URL: https://www.rfc-editor.org/info/rfc8017 (Last accessed: 13.04.2024).

14) Viega J., McGrew D. The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP). Request for Comments. № 4106. RFC Editor, 2005. 11 p. URL: https://www.rfc-editor.org/info/rfc4106 (Last accessed: 14.04.2024).