МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА СЕРВІСУ ДЛЯ СПІЛКУВАННЯ ТА ОБМІНУ ДАНИМИ З ВИКОРИСТАННЯМ DJANGO»**

Виконав: студент 4 курсу, групи 6.1211-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.О. Квітко
(ініціали та прізвище)
Керівник доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н. Панасенко Є.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Квітку Максиму Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу для спілкування та обміну даними з використанням Django

керівник роботи Панасенко Євген Валерійович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Проєктування сервісу.
3. Програмна реалізація та тестування сервісу.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.01.2025	
2.	Збір вихідних даних.	20.01.2025	
3.	Обробка методичних та теоретичних	11.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	21.03.2025	
5.	Розробка третього розділу.	12.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ М.О. Квітко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Є.В. Панасенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка сервісу для спілкування та обміну даними з використанням Django»: 52 с., 28 рис., 1 табл., 10 джерел.

АСИНХРОННЕ ПРОГРАМУВАННЯ, БАЗА ДАНИХ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ОБМІН ДАНИМИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СПІЛКУВАННЯ, DJANGO, PYQT, PYTHON, WEBSOCKET.

Об’єкт дослідження – процес розробки та функціонування програмного сервісу.

Мета роботи: розробка сервісу, для спілкування та обміну даними між користувачами засобами Django, PyQt та Websocket.

Метод дослідження – методи аналізу до вимог програмного забезпечення, методи моделювання, тестування, проєктування та конструювання програмного забезпечення.

У кваліфікаційній роботі досліджено та реалізовано процес створення програмного сервісу, призначеного для спілкування та обміну даними. Основна увага приділена технічній реалізації взаємодії між серверною частиною, розробленою на фреймворку Django, та клієнтським настільним застосунком, створеним за допомогою бібліотеки PyQt. Для забезпечення обміну інформацією в режимі реального часу між сервером та клієнтом використано технологію WebSocket.

В ході роботи було проаналізовано особливості обраних технологій, спроєктовано архітектуру системи, що включає серверну та клієнтську компоненти, та розроблено основні функціональні модулі. Результатом роботи є програмний прототип, що демонструє можливість ефективної інтеграції Django, PyQt та WebSocket для створення сервісів, орієнтованих на обмін даними та комунікацію. Робота зосереджена на технічних аспектах реалізації ядра такого сервісу.

SUMMARY

Bachelor’s qualifying paper “Development of a Service for Communication and Data Exchange Using Django”: 52 pages, 28 figures, 1 table, 10 references.

ASYNCHRONOUS PROGRAMMING, DATABASE, CLIENT-SERVER ARCHITECTURE, DATA EXCHANGE, SOFTWARE, COMMUNICATION, DJANGO, PYQT, PYTHON, WEBSOCKET.

The object of the study is the process of developing and functioning a software service.

The aim of the study is to develop a service for communication and data exchange between users using Django, PyQt, and Websocket tools.

The methods of research are methods of analysis to software requirements, methods of modeling, testing, software design and construction.

In the qualification work, the process of creating a software service designed for communication and data exchange is studied and implemented. The main focus is on the technical implementation of interaction between the server part developed on the Django framework and the client desktop application created using the PyQt library. WebSocket technology is used to ensure real-time information exchange between the server and the client.

In the course of the work, the features of the selected technologies were analyzed, the system architecture was designed, which includes server and client components, and the main functional modules were developed. The result is a software prototype that demonstrates the ability to effectively integrate Django, PyQt, and WebSocket to create services focused on data exchange and communication. The work focuses on the technical aspects of implementing the core of such a service.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Технічне завдання 10
1.1 Терміни та визначення..... 10
1.1.1 Загальні терміни..... 10
1.1.2 Технічні терміни 11
1.2 Функціональні вимоги..... 11
1.2.1 Призначення і цілі створення системи 11
1.2.2 Загальні функціональні можливості системи 12
1.3 Нефункціональні вимоги..... 12
1.3.1 Інтерфейс користувача 12
1.3.2 Підтримка операційних систем 13
1.3.3 Вимоги до продуктивності..... 13
1.3.4 Вимоги до безпеки..... 13
1.4 Опис предметної області 14
1.5 Опис системи..... 14
1.6 Порівняння з подібними системами..... 15
2 Проєктування..... 19
2.1 Використання UML під час розробки системи..... 19
2.2 Діаграма варіантів використання 20
2.2.1 Опис варіантів використання..... 21
3 Реалізація та тестування 25
3.1 Опис інструментів розробки 25
3.2 Структура системи..... 26
3.2.1 Клієнтська частина 26
3.2.2 Серверна частина 35

3.3 Тестування 41
3.3.1 Методологія тестування 41
3.3.2 Сценарії тестування та результати 41
Висновки 50
Перелік посилань 52

ВСТУП

У цифрову епоху, коли інформація та комунікація стають рушійною силою прогресу, потреба в надійних та гнучких інструментах для взаємодії людей та обміну даними лише зростає. Сучасні користувачі очікують не просто можливості обмінюватися повідомленнями, а й швидкості, безпеки та зручності, незалежно від платформи чи пристрою. Саме тому розробка нових сервісів спілкування, що враховують специфічні потреби та використовують передові технології, залишається надзвичайно актуальним завданням.

Дана кваліфікаційна робота присвячена дослідженню та практичній реалізації такого сервісу – багатофункціональної платформи для спілкування та обміну даними. Особливістю проєкту є поєднання потужного вебфреймворку Django для розробки стабільної та масштабованої серверної частини, технології WebSocket для забезпечення миттєвого обміну повідомленнями в режимі реального часу, та бібліотеки PyQt для створення зручного та функціонального настільного клієнтського додатку. Такий технологічний стек дозволяє створити комплексне рішення, що забезпечує як надійність серверної інфраструктури, так і комфорт користувацької взаємодії.

Розроблюваний сервіс покликаний вирішити проблему ефективної організації комунікації, надаючи користувачам інструменти для обміну текстовими повідомленнями, файлами та іншими даними у захищеному та контрольованому середовищі. На відміну від багатьох існуючих рішень, акцент робиться на створенні самостійної платформи з гнучкими можливостями кастомізації та потенційним розширенням функціоналу.

Для реалізації поставленої мети, а саме розробки сервісу для спілкування та обміну даними засобами Django, PyQt та WebSocket, у роботі визначено наступні ключові завдання:

– провести аналіз сучасних підходів та технологій для створення комунікаційних платформ;

- спроєктувати архітектуру майбутнього сервісу, визначивши ключові компоненти та взаємозв’язки між ними;
- розробити серверну частину на базі фреймворку Django, включаючи моделі даних, API для взаємодії з клієнтом та логіку обробки повідомлень;
- імплементувати механізм обміну даними в реальному часі за допомогою технології WebSocket;
- створити настільний клієнтський додаток з інтуїтивно зрозумілим графічним інтерфейсом, використовуючи бібліотеку PyQt;
- забезпечити інтеграцію серверної та клієнтської частин, провести тестування розробленого функціоналу та оцінити його працездатність.

Предметом дослідження в рамках даної кваліфікаційної роботи є специфічні методи, інструменти та підходи до створення сервісу комунікації, що базується на синергії технологій Django, PyQt та WebSocket, а також особливості їх інтеграції для досягнення поставлених функціональних вимог.

Робота складається зі вступу, трьох основних розділів, що послідовно розкривають теоретичні основи, етапи проєктування та практичну реалізацію сервісу, а також висновків та списку використаних джерел. Перший розділ присвячений огляду предметної області та обґрунтуванню вибору технологічного стеку. Другий розділ детально описує архітектуру системи та проєктні рішення. Третій розділ демонструє процес розробки ключових модулів сервісу та результати його тестування.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – програмний комплекс «Сервіс для спілкування та обміну даними», що розробляється.

Користувач – особа, яка зареєстрована в системі та використовує її функціонал для спілкування та обміну даними.

Адміністратор – користувач з розширеними правами, який має доступ до функцій керування користувачами та іншими аспектами системи.

Обліковий запис – сукупність даних про користувача (логін, пароль, ім’я, список друзів тощо), необхідних для його ідентифікації та авторизації в системі.

Чат – віртуальна кімната для обміну повідомленнями та файлами між двома або більше користувачами.

Повідомлення – текстова інформація, що передається між користувачами в межах чату.

Файл – дані будь-якого формату (зображення, документ, архів тощо), що передаються між користувачами в межах чату.

Список друзів – перелік користувачів, з якими поточний користувач встановив зв’язок для спрощеного доступу до спілкування.

Авторизація – процес перевірки прав користувача на доступ до системи шляхом введення логіна та пароля.

Реєстрація – процес створення нового облікового запису в системі.

Клієнтський додаток (Клієнт) – частина системи, що встановлюється на пристрій користувача (реалізована на PyQt), через яку він взаємодіє з серверною частиною.

Серверна частина (Сервер) – частина системи (реалізована на Django), що обробляє запити від клієнтів, зберігає дані та керує логікою взаємодії.

1.1.2 Технічні терміни

Django – високорівневий Python вебфреймворк, що використовується для розробки серверної частини системи [1].

PyQt – набір прив’язок Python до Qt, бібліотеки для створення графічного інтерфейсу користувача (GUI), що використовується для розробки клієнтського додатку [3].

WebSocket – протокол зв’язку, що забезпечує двосторонній обмін даними між клієнтом та сервером в режимі реального часу [4].

База даних (БД) – структуроване сховище даних системи (наприклад, інформація про користувачів, повідомлення, чати).

API (Application Programming Interface) – набір правил та протоколів, за допомогою яких клієнт взаємодіє з сервером.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість обміну текстовими повідомленнями та файлами між зареєстрованими користувачами в режимі реального часу, а також керування списком контактів (друзів).

Експлуатаційне призначення системи: система може експлуатуватися користувачами для особистого та групового спілкування, а також адміністратором для керування обліковими записами користувачів.

Мета створення системи – розробка настільного додатку-месенджера для зручного та швидкого спілкування та обміну даними.

1.2.2 Загальні функціональні можливості системи

Система має надавати користувачам такі можливості:

– створення нового облікового запису (реєстрація);
– вхід до системи за допомогою логіна та пароля (авторизація);
– вихід із системи (логаут);
– перегляд списку наявних чатів;
– вибір чату для спілкування;
– створення нового чату з іншим користувачем;
– надсилання текстових повідомлень у вибраний чат;
– надсилання файлів у вибраний чат.

Система має надавати адміністраторам такі можливості:

– вхід до системи з правами адміністратора;
– перегляд списку всіх зареєстрованих користувачів;
– редагування даних користувачів;
– блокування/розблокування облікових записів користувачів;
– видалення облікових записів користувачів.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Клієнтський додаток повинен мати інтуїтивно зрозумілий та зручний графічний інтерфейс.

Інтерфейс повинен коректно відображатися на підтримуваних операційних системах та роздільних здатностях екрану.

Елементи керування повинні бути чітко ідентифіковані та доступні.

1.3.2 Підтримка операційних систем

Клієнтський додаток Системи повинен коректно працювати на наступних операційних системах:

– Windows (версія 10 та новіші);
– Linux (наприклад, Ubuntu 20.04 LTS та новіші).

1.3.3 Вимоги до продуктивності

Час відгуку інтерфейсу на дії користувача (наприклад, відкриття вікна, натискання кнопки) не повинен перевищувати 1 секунду.

Доставка текстового повідомлення до адресата (за умови стабільного інтернет-з’єднання) не повинна перевищувати 2-3 секунди.

Завантаження списку чатів або списку друзів (до 100 записів) не повинно тривати довше 2 секунд.

Система повинна стабільно працювати при одночасній активності до хоча б 50 користувачів.

1.3.4 Вимоги до безпеки

Паролі користувачів повинні зберігатися в базі даних у хешованому вигляді.

Система не повинна надавати доступ до функціоналу неавторизованим користувачам, за винятком форми реєстрації та входу.

Передача даних між клієнтом та сервером повинна здійснюватися захищеним каналом (наприклад, з використанням TLS/SSL для WebSocket [4]).

Система повинна передбачати захист від базових атак.

Доступ до адміністративного інтерфейсу повинен бути суворо обмежений для користувачів з роллю адміністратора.

1.4 Опис предметної області

Предметною областю є розробка сервісу для забезпечення комунікації між користувачами в режимі реального часу. Сучасні потреби вимагають швидкого та зручного обміну інформацією, включаючи не тільки текстові повідомлення, але й файли різного типу. Дана система покликана задовольнити цю потребу шляхом надання користувачам настільного додатку-месенджера.

Процес взаємодії з системою починається із запуску клієнтського додатку. Користувач повинен ввести свої облікові дані (логін та пароль) у форму входу. Якщо користувач використовує систему вперше, він повинен пройти процедуру реєстрації, вказавши необхідні дані для створення нового облікового запису.

Після успішної авторизації користувач отримує доступ до основного інтерфейсу, де він може бачити список своїх чатів або список друзів. Користувач може обрати існуючий чат для продовження спілкування або створити новий чат з іншим користувачем системи (наприклад, знайшовши його через функцію пошуку та безпосередньо ініціювавши чат). У вибраному чаті користувач може надсилати текстові повідомлення та прикріплювати файли. Всі повідомлення та файли відображаються в хронологічному порядку. Користувач також може вийти зі свого облікового запису.

Адміністратор системи, після авторизації з відповідними правами, має доступ до функцій керування користувачами, таких як перегляд списку користувачів, їх блокування чи видалення та зміну їх даних при винятках.

1.5 Опис системи

Спілкування є невід’ємною частиною повсякденного життя та роботи. Існуючі рішення (публічні месенджери, електронна пошта) не завжди задовольняють специфічні потреби або можуть мати обмеження щодо

приватності чи функціоналу. Розробка власного сервісу для спілкування дозволяє створити контрольоване середовище для обміну інформацією.

Система «Сервіс для спілкування та обміну даними» є клієнт-серверним додатком. Клієнтська частина, розроблена на PyQt [3], надає графічний інтерфейс для взаємодії користувача. Серверна частина, реалізована на Django [1], відповідає за обробку логіки, збереження даних (користувачі, чати, повідомлення, файли) у базі даних та забезпечення зв’язку в реальному часі за допомогою технології WebSocket [4].

Можливості системи:

а) основні для користувача:
   1) реєстрація та авторизація;
   2) керування списком чатів та списком друзів;
   3) обмін текстовими повідомленнями в режимі реального часу;
   4) обмін файлами;

б) консоль адміністратора (якщо реалізується):
   1) керування обліковими записами користувачів.

1.6 Порівняння з подібними системами

Для оцінки позиціонування та функціональних особливостей розроблюваного «Сервісу для спілкування та обміну даними» проведемо порівняльний аналіз з популярними на ринку месенджерами: Telegram та Discord (табл. 1.1).

Таблиця 1.1 – Порівняння проєкту з подібними системами

Характеристика	Telegram Desktop	Discord	Розроблювана Система
Основне призначення	Швидкий та безпечний обмін повідомленнями, файлами, медіа; канали, групи, боти	Спілкування для спільнот (особливо геймерських), голосові та відеочати, стрімінг	Базовий обмін текстовими повідомленнями та файлами між користувачами

Продовження таблиці 1.1

Характеристика	Telegram Desktop	Discord	Розроблювана Система
Платформи	Windows, macOS, Linux, Android, iOS, Web	Windows, macOS, Linux, Android, iOS, Web	Windows, Linux
Основні технології	Клієнт: C++, Qt; Сервер: C++, власні протоколи (MTProto)	Клієнт: Electron (JavaScript, HTML, CSS); Сервер: Elixir, Rust, Go, Python	Клієнт: Python (PyQt); Сервер: Python (Django), WebSocket
Типи чатів	Приватні, групові чати, супергрупи, канали	Приватні, групові чати, текстові та голосові канали на серверах	Приватні чати
Обмін файлами	Так, великі файли (до 2 ГБ), будь-які типи	Так, обмеження розміру залежить від типу підписки (8 МБ безкоштовно)	Так, типи та розмір файлів будуть обмежені конфігурацією сервера
Голосові/Відео дзвінки	Так (приватні та групові)	Так (приватні, групові, на серверах)	Ні (не заплановано)
Боти та API	Потужна платформа для ботів, відкрите API	Широкі можливості для ботів, відкрите API	Ні (не заплановано)
Кастомізація інтерфейсу	Обмежена (теми)	Розширена (теми, плагіни через сторонні інструменти)	Базова
Безпека (основні аспекти)	Клієнт-серверне шифрування, опціональні секретні чати (E2EE)	Клієнт-серверне шифрування, двофакторна автентифікація	Клієнт-серверне шифрування, хешування паролів
Монетизація	Premium-підписка, реклама в каналах	Premium-підписка (Nitro), продаж ігор	Відсутня (навчальний проєкт)
Цільова аудиторія	Широка аудиторія, акцент на приватності та швидкості	Спільноти, геймери, команди	Користувачі, яким потрібен простий інструмент для обміну повідомленнями та файлами в контрольованому середовищі

Переваги розроблюваної системи (у контексті навчального проєкту та потенційного нішевого використання):

– **поглиблене вивчення технологій**: проєкт дозволяє отримати практичний досвід розробки клієнт-серверного додатку з

використанням стеку Python: Django для бекенду [1], PyQt для десктопного клієнту [3] та WebSocket для комунікації в реальному часі [4];

- **контроль над середовищем**: як власний розгортуваний додаток, система надає повний контроль над серверною частиною, даними користувачів та логікою роботи, що може бути важливим для специфічних завдань або невеликих закритих груп;
- **простота та мінімалізм**: система фокусується на базовому функціоналі обміну повідомленнями та файлами, що робить її потенційно легкою та невимогливою до ресурсів, а також простою у використанні для користувачів, яким не потрібні розширені функції великих платформ;
- **можливість кастомізації**: будучи власним проєктом, система може бути легко модифікована та розширена в майбутньому для задоволення конкретних потреб, які не покриваються стандартними месенджерами.

Недоліки розроблюваної системи (у порівнянні з Telegram та Discord):

- **значно обмежений функціонал**: відсутність голосових/відеодзвінків, розширених можливостей для групового спілкування (складні ролі, модерація на серверах Discord), ботів, стікерів, інтеграцій з іншими сервісами;
- **відсутність мобільних та вебклієнтів**: доступність лише через десктопний додаток на обмеженому переліку ОС суттєво звужує коло потенційних користувачів та сценаріїв використання;
- **менший рівень безпеки та надійності**: як навчальний проєкт, система, ймовірно, не матиме такого рівня тестування, оптимізації та захисту від загроз, як продукти, що розробляються великими командами з відповідними ресурсами;
- **складність залучення користувачів**: відсутність ефекту мережі та відомого бренду робить залучення користувачів складним завданням поза межами конкретних вузьких ніш або навчальних цілей;

– **відсутність E2EE (End-to-End Encryption) за замовчуванням**: популярні месенджери приділяють значну увагу питанням наскрізного шифрування, реалізація якого в даному проєкті може бути складною.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

Застосування UML (Unified Modeling Language) у процесі розробки системи відіграє важливу роль у ефективному проєктуванні та реалізації програмного забезпечення.

UML забезпечує уніфікований набір графічних інструментів і нотацій, які дозволяють розробникам чітко уявити, описати та змоделювати систему ще до початку її впровадження.

У контексті розробки сервісу для спілкування та обміну даними, що використовує Django для серверної частини [1], PyQt6 для клієнтського застосунку [3] та Websocket для обміну повідомленнями в реальному часі [4], UML дозволяє чітко візуалізувати взаємодії між користувачами та системою, структуру компонентів та потоки даних.

Застосування UML сприяє виявленню потенційних проблем та вузьких місць на ранніх етапах розробки, що дозволяє зекономити час та ресурси. Моделювання архітектури системи, взаємодії її компонентів та поведінки користувачів допомагає забезпечити, що кінцевий продукт буде відповідати функціональним та нефункціональним вимогам.

Уміння користуватись UML також сприяє покращенню комунікації між командою розробників та, за потреби, замовником або іншими зацікавленими сторонами.

Графічне представлення системи у вигляді діаграм дозволяє краще зрозуміти, яким чином система буде функціонувати та які можливості вона надає. Це сприяє виявленню можливих розбіжностей між очікуваннями та побудованою системою ще на ранніх етапах розробки.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) є важливим елементом UML, що слугує для моделювання та аналізу взаємодії системи з зовнішніми користувачами або іншими системами. Вона графічно зображає акторів і варіанти використання, які ілюструють можливі сценарії взаємодії між ними та системою. Для сервісу спілкування та обміну даними діаграма варіантів використання допомагає визначити основні функції, доступні різним типам користувачів, та загальну логіку їх взаємодії з системою.

На рисунку 2.1 представлена діаграма варіантів використання розробленого сервісу.

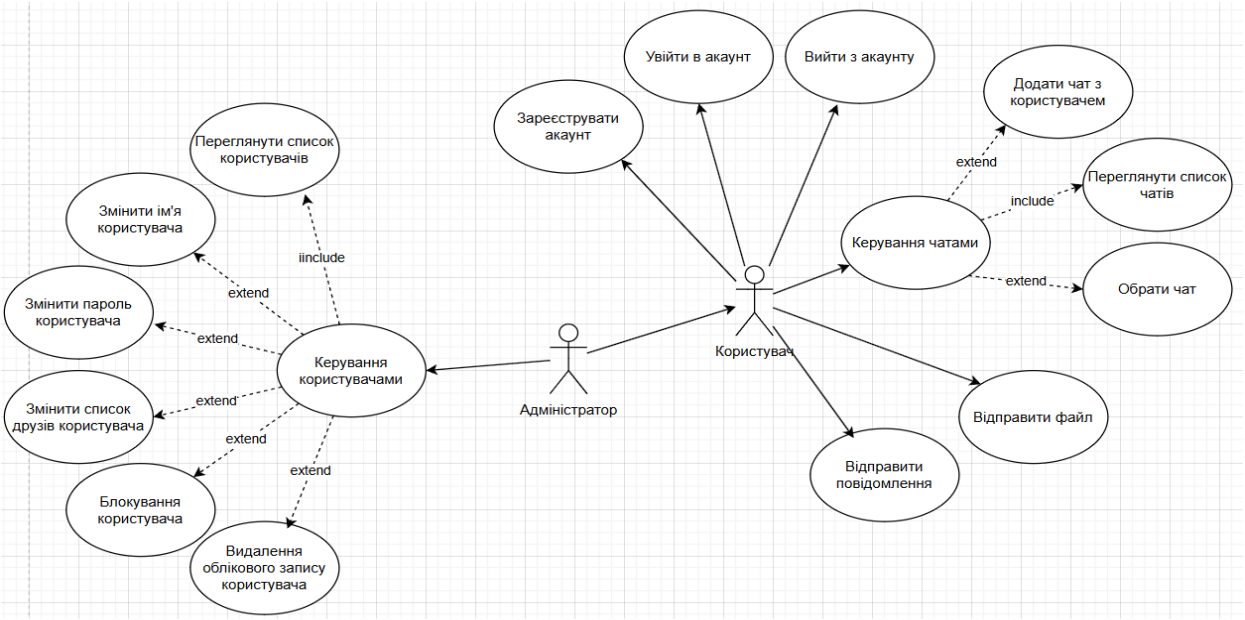


Рисунок 2.1 – Діаграма варіантів використання

На діаграмі представлено два основних актори:

– **«Користувач»:** представляє типового кінцевого користувача системи, який може реєструватися, входити в систему, створювати чати, обмінюватися повідомленнями та файлами;

– **«Адміністратор»:** є користувачем з розширеними правами, який успадковує всі можливості «Користувача» та додатково може керувати обліковими записами інших користувачів.

Виділено декілька ключових груп варіантів використання. Для «Користувача» це взаємодія з акаунтом (реєстрація, вхід, вихід), керування чатами (створення, вибір, перегляд списку) та обмін даними (надсилання повідомлень та файлів). Для «Адміністратора» основною додатковою функціональністю є «Керування користувачами», що включає перегляд списку користувачів, зміну їхніх даних, блокування та видалення акаунтів.

2.2.1 Опис варіантів використання

Прецедент «Зареєструвати акаунт».

Призначення: даний варіант використання надає можливість новому користувачу створити обліковий запис у системі для подальшого використання застосунку.

Основний потік подій: користувач обирає опцію реєстрації. Система відображає форму для введення необхідних даних (логін, пароль). Після заповнення форми та підтвердження, система перевіряє унікальність логіна та валідність даних, створює новий обліковий запис користувача та повідомляє про успішну реєстрацію.

Альтернативний потік подій: якщо користувач вже існує або введені дані невалідні, система повідомляє про помилку та пропонує виправити дані або обрати інший логін.

Передумова: користувач не автентифікований у системі.

Прецедент «Увійти в акаунт».

Призначення: даний варіант використання дозволяє зареєстрованому користувачу отримати доступ до свого облікового запису та функціоналу системи.

Основний потік подій: користувач вводить свої облікові дані (логін та пароль) у відповідну форму. Система перевіряє їх коректність. У разі успішної автентифікації користувач отримує доступ до основного інтерфейсу сервісу.

Альтернативний потік подій: якщо введені облікові дані невірні, система повідомляє про помилку та пропонує повторити спробу.

Передумова: користувач має зареєстрований обліковий запис.

Прецедент «Керування чатами» *(загальний прецедент для групування дій над чатами для «Користувача»)*.

Призначення: даний варіант використання надає користувачу можливість створювати нові чати, обирати існуючі для спілкування та переглядати список доступних чатів.

Основний потік подій: після успішного входу в систему, користувач може взаємодіяти з функціоналом чатів. Це включає:

- **«Додати чат з користувачем»**: користувач ініціює створення нового чату, обираючи іншого користувача системи;
- **«Обрати чат»**: користувач вибирає існуючий чат зі списку для перегляду історії повідомлень та надсилання нових;
- **«Переглянути список чатів»**: система відображає список чатів, у яких бере участь користувач.

Передумова: користувач повинен виконати вхід до системи.

Прецедент «Відправити повідомлення».

Призначення: даний варіант використання дозволяє користувачу надсилати текстові повідомлення в обраний чат.

Основний потік подій: користувач обирає активний чат, вводить текст повідомлення у відповідне поле та натискає кнопку відправки. Система, використовуючи Websocket [4], передає повідомлення іншим учасникам чату в реальному часі. Повідомлення зберігається в історії чату.

Передумова: користувач повинен виконати вхід до системи та обрати активний чат.

Виняткова ситуація: не вдалося встановити з’єднання через Websocket [4] – система повідомляє про помилку надсилання.

Прецедент «Відправити файл».

Призначення: даний варіант використання дозволяє користувачу надсилати файли в обраний чат.

Основний потік подій: користувач обирає активний чат, натискає кнопку для прикріплення файлу, обирає файл зі свого пристрою та підтверджує відправку. Система завантажує файл на сервер та надсилає його іншим учасникам чату. Файл стає доступним для завантаження.

Передумова: користувач повинен виконати вхід до системи та обрати активний чат.

Виняткова ситуація 1: файл перевищує дозволений розмір або має неприпустимий тип – система повідомляє про помилку.

Виняткова ситуація 2: помилка завантаження файлу на сервер – система повідомляє про помилку.

Прецедент «Керування користувачами» *(загальний прецедент для «Адміністратора»)*.

Призначення: даний варіант використання надає адміністратору можливість керувати обліковими записами користувачів системи.

Основний потік подій: адміністратор, після автентифікації з відповідними правами, отримує доступ до панелі керування користувачами. Система відображає список користувачів (виконується прецедент «Переглянути список користувачів», який має зв’язок “include”). Адміністратор може обрати користувача та виконати над ним одну з розширюючих дій (зв’язок “extend”): змінити ім’я, змінити пароль, змінити список друзів (або інші профільні дані), заблокувати користувача або видалити його обліковий запис.

Альтернативний потік подій: якщо в системі немає зареєстрованих користувачів (окрім самого адміністратора), система сповістить про це у вигляді напису «Немає жодного запису для керування».

Передумова: адміністратор повинен виконати вхід до системи з правами адміністратора.

Прецедент «Переглянути список користувачів» *(прецедент, який має зв’язок “include” з батьківським «Керування користувачами»)*.

Призначення: даний варіант використання надає адміністратору можливість переглядати список всіх зареєстрованих користувачів системи.

Основний потік подій: даний варіант використання автоматично виконується, коли адміністратор переходить до розділу керування користувачами. Система запитує з бази даних та відображає список користувачів із зазначенням їх основних даних.

Альтернативний потік подій: якщо в системі немає зареєстрованих користувачів, система сповістить про це.

Передумова: адміністратор повинен виконати вхід до системи та перейти до розділу керування користувачами.

Прецедент «Видалення облікового запису користувача» *(прецедент, який має зв'язок "extend" з батьківським «Керування користувачами»)*.

Призначення: даний варіант використання надає адміністратору можливість видалити обліковий запис користувача з системи.

Основний потік подій: адміністратор обирає користувача зі списку та натискає кнопку «Видалити». Система запитує підтвердження дії. Після підтвердження, система видаляє всі дані, пов'язані з обліковим записом користувача, та повідомляє адміністратора про успішне видалення.

Передумова: адміністратор повинен знаходитися в розділі «Керування користувачами» та обрати конкретного користувача для видалення.

Виняткова ситуація 1: спроба видалити власний обліковий запис адміністратора – система блокує дію та відображає відповідне повідомлення.

Виняткова ситуація 2: адміністратор скасовує дію видалення – система повертається до списку користувачів.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для розробки сервісу спілкування та обміну даними було обрано набір сучасних та ефективних інструментів і технологій, що дозволяють реалізувати як серверну, так і клієнтську частини системи, а також забезпечити обмін даними в реальному часі.

Django – високорівневий Python вебфреймворк, що використовується для розробки серверної частини (backend) [1]. Django дозволяє швидко розробляти надійні та масштабовані вебдодатки, надаючи інструменти для роботи з базами даних (ORM), маршрутизації запитів, автентифікації користувачів, створення REST API (за допомогою Django REST framework) та адміністрування системи.

Django REST framework (DRF) – потужний та гнучкий інструментарій для створення Web API на базі Django. Використовується для реалізації ендпоінтів, через які клієнтська частина взаємодіє з сервером для операцій, що не потребують реального часу (наприклад, реєстрація, автентифікація, отримання списку чатів/користувачів) [5].

Django Channels – розширення для Django, що додає підтримку протоколів, відмінних від HTTP [2], зокрема WebSocket [4]. Використовується для організації двостороннього зв’язку в реальному часі між клієнтом та сервером, що є критично важливим для функціоналу обміну миттєвими повідомленнями.

PyQt6 – набір Python-бібліотек для кросплатформенного фреймворку Qt6 [3, 7, 10]. Використовується для розробки графічного інтерфейсу користувача (GUI) клієнтського застосунку. PyQt6 дозволяє створювати нативні та функціональні десктопні додатки з багатим набором віджетів та можливостей.

WebSockets – протокол зв’язку, що забезпечує повнодуплексний канал зв’язку через одне TCP-з’єднання [4]. Використовується для миттєвої передачі повідомлень та файлів між клієнтами через сервер на базі Django Channels [2, 9].

SQLite – система управління базами даних. Для розробки та тестування може використовуватися SQLite через її простоту інтеграції з Django.

3.2 Структура системи

Розроблений сервіс для спілкування та обміну даними побудований за клієнт-серверною архітектурою. Клієнтська частина, що відповідає за взаємодію з користувачем, реалізована з використанням бібліотеки PyQt6. Серверна частина, що обробляє логіку застосунку, побудована на базі фреймворку Django [1] з використанням Django Channels [2] для забезпечення обміну даними в реальному часі через протокол WebSocket [4]. Дані чатів (історія повідомлень та файли) зберігаються у файловій системі сервера.

3.2.1 Клієнтська частина

Клієнтська частина відповідає за відображення графічного інтерфейсу, обробку дій користувача, взаємодію з сервером через WebSocket [4] для обміну повідомленнями, файлами та іншою інформацією.

Клас MainWindow(QMainWindow). Цей клас є основним вікном клієнтського застосунку. Він керує відображенням усіх елементів інтерфейсу, таких як список друзів, область чату, поля для вводу повідомлень, а також обробляє події, пов’язані з діями користувача [8].

Призначення:

– ініціалізація та відображення головного вікна програми;

– керування станами інтерфейсу (екран входу, основний екран чату);

– відображення списку друзів та лічильників непрочитаних повідомлень;
– відображення історії повідомлень обраного чату;
– обробка відправлення та отримання текстових повідомлень і файлів;
– керування процесом входу та виходу користувача;
– взаємодія з класом WebsocketClient для комунікації з сервером.

Головні методи.

__init__(self): конструктор класу, ініціалізує початкові параметри вікна та відображає віджет для входу (LoginWidget).

init_ui(self): ініціалізує основний інтерфейс користувача після успішного входу. Створює та розміщує панель друзів, область для повідомлень, кнопки меню (додати друга, вийти) [7, 10] (рис. 3.1).

```
def init_ui(self):
    self.main_widget = QWidget()
    self.main_layout = QVBoxLayout()
    # ... (створення кнопок меню) ...
    menu_layout.addWidget(add_friend_button, alignment=Qt.AlignmentFlag.AlignLeft)
    menu_layout.addWidget(logout_button, alignment=Qt.AlignmentFlag.AlignRight)
    self.main_layout.addLayout(menu_layout)

    self.friends_panel = QHBoxLayout()
    # ... (завантаження списку друзів) ...
    asyncio.create_task(self.load_friends())
    self.main_layout.addLayout(self.friends_panel)

    # ... (підключення до лічильника непрочитаних повідомлень) ...
    asyncio.create_task(self.unread_client.connect_unread_counter(self.username))

    self.scroll_area = QScrollArea(objectName="messages_scroll_area")
    # ... (налаштування області чату) ...
    self.main_layout.addWidget(self.scroll_area)
    self.setCentralWidget(self.main_widget)
    return self.main_widget
```

Рисунок 3.1 – Частковий код методу ініціалізації графічного інтерфейсу класу MainWindow

on_message_received(self, data): асинхронно обробляє повідомлення, отримані від WebSocket сервера. Це можуть бути як окремі нові повідомлення, так і історія чату (рис. 3.2).

```
async def on_message_received(self, data):
    if 'history' in data:
        self.setUpdatesEnabled(False)
        self.clear_chat()
        # ... (обробка історії повідомлень) ...
        for index, msg in enumerate(data['history']):
            if 'filename' in msg:
                self.display_message(msg['sender'], msg['filename'].split('_', 1)[1], msg['timestamp'], is_file=True,
full_filename=msg['filename'])
            else:
                self.display_message(msg['sender'], msg['message'], msg['timestamp'])
            await asyncio.sleep(0.01)
            await self.check_unread_messages(msg['sender'], index, msg['is_read'], self.previous_msg_pos)
        # ... (прокрутка чата та оновлення лічильника непрочитаних повідомлень) ...
        self.setUpdatesEnabled(True)
    else: # Отримано нове повідомлення
        if 'filename' in data:
            self.display_message(data['sender'], data['filename'].split('_', 1)[1], data['timestamp'], is_file=True,
full_filename=data['filename'])
        else:
            self.display_message(data['sender'], data['message_text'], data['timestamp'])
        # ... (оновлення лічильника непрочитаних повідомлень та прокрутка чата на позицію першого такого
повідомлення) ...
```

Рисунок 3.2 – Частковий код методу обробки повідомлень з сервера на клієнт класу MainWindow

display_message(self, sender, message_text, timestamp, is_file=False, full_filename=None): формує та відображає окреме повідомлення (текстове або файл) в інтерфейсі чату, враховуючи відправника.

load_friends(self): асинхронно завантажує список друзів користувача з сервера та відображає їх на панелі друзів.

switch_chat(self, username, friend_name): асинхронно обробляє перемикання на чат з обраним другом. Ініціює закриття попереднього WebSocket з’єднання (якщо було) та відкриття нового для обраного чату, а також завантаження історії повідомлень (рис. 3.3).

```
async def switch_chat(self, username, friend_name):
    usernames = sorted([username, friend_name])
    self.chat_name = f"{usernames[0]}_{urllib.parse.quote(usernames[1])}"

    if self.chat_client and self.chat_client.chat_name != self.chat_name:
        await self.chat_client.disconnect_chat()

    self.chat_client = WebsocketClient(self)
    await self.chat_client.connect_chat(self.chat_name)
    self.add_message_input_widgets()
```

Рисунок 3.3 – Код методу підключення до чату для подальшого спілкування/відправки файлів класу MainWindow

send_message(self): асинхронно відправляє текстове повідомлення, введене користувачем, на сервер через WebsocketClient (рис. 3.4).

```
async def send_message(self):
    message = self.message_input.toPlainText().strip()
    if message:
        await self.chat_client.send_chat_message_or_file(
            is_file=False,
            sender=self.username,
            message=message
        )
        self.message_input.clear()
```

Рисунок 3.4 – Код методу відправки повідомлення користувача для обробки на сервер класу MainWindow

send_file(self, file_path): асинхронно обробляє відправку обраного файлу: читає файл, кодує його в Base64 та відправляє на сервер (рис. 3.5).

```
async def send_file(self, file_path):
    filename = os.path.basename(file_path)
    with open(file_path, 'rb') as f:
        file_content = f.read()

        # Encoded in Base64 and decoded to UTF-8 string for JSON
        encoded_content = base64.b64encode(file_content).decode('utf-8')
        print(f"File '{filename}' encoded to Base64.")

        if self.chat_client:
            clean_chat_name = self.chat_name
            await self.chat_client.send_chat_message_or_file(
                is_file=True,
                sender=self.username,
                filename=filename,
                filedata=encoded_content
            )
```

Рисунок 3.5 – Код методу відправки попередньо закодованого файлу на сервер класу MainWindow

handle_file_download(self, filename_to_download): асинхронно обробляє запит на завантаження файлу з сервера, отримує дані файлу та зберігає їх (рис. 3.6).

```
async def handle_file_download(self, filename_to_download):
    if not self.chat_client or not self.chat_name:
        print("Download Error", "Not connected to a chat to download file.")
        return

    save_path, _ = QFileDialog.getSaveFileName(self, "Save File As...",
filename_to_download.split('_', 1)[1])
    if not save_path:
        print("File save cancelled by user.")
        return

    response = await WebsocketClient.send_request(
            endpoint="chat/",
            request_type="download_file",
            data={
                "chat_name": self.chat_name,
                "file_name": filename_to_download
            }
        )

    if response['status'] == "success":
        encoded_content = response['file_data_base64']
        file_content_binary = base64.b64decode(encoded_content)
        with open(save_path, 'wb') as f:
                f.write(file_content_binary)
        print(f"File successfully saved to {save_path}")
```

Рисунок 3.6 – Код методу завантаження файла з чату класу MainWindow

on_update_unread_messages_count(self, data): оновлює лічильник непрочитаних повідомлень для відповідного друга в списку.

on_logout(self): асинхронно обробляє вихід користувача із системи, закриває WebSocket з’єднання та повертає користувача на екран входу.

Клас LoginWidget().

Призначення: надає користувацький інтерфейс [7, 10] для введення логіна та пароля. Ініціює процес автентифікації через WebsocketClient та керує переходом до основного інтерфейсу або до віджета реєстрації.

Головні методи.

__init__(self, main_window): конструктор, ініціалізує віджет та зберігає посилання на головне вікно.

init_ui(self): створює та повертає віджет з полями для вводу імені користувача, пароля, кнопками “Login” та “Register”, а також міткою для відображення попереджень.

on_login(self): асинхронний метод, який викликається при натисканні кнопки “Login”. Збирає введені дані, відправляє запит на сервер через WebsocketClient.send_request з endpoint="login/". У разі успіху викликає login_successful(), інакше відображає повідомлення про помилку (рис. 3.7).

```
async def on_login(self):
    username = self.username_input.text().strip()
    password = self.password_input.text()

    if not username or not password:
        self.warning_label.setText("All fields are required!")
        # ... (відображення попередження) ...
        return

    response = await WebsocketClient.send_request(
        endpoint="login/",
        request_type="login",
        data={"username": username, "password": password}
    )

    if response['status'] == 'success':
        self.main_window.username = username
        self.login_successful()
    else:
        self.warning_label.setText(response['message'])
        # ... (відображення попередження) ...
```

Рисунок 3.7 – Код методу відправки на сервер для перевірки введених користувачем даних авторизації класу LoginWidget

login_successful(self): перемикає центральний віджет головного вікна на основний інтерфейс користувача, викликаючи self.main_window.init_ui().

on_go_to_register(self): викликається при натисканні кнопки “Register”. Створює екземпляр RegistrationWidget та встановлює його як центральний віджет головного вікна.

Клас RegistrationWidget().

Призначення: надає користувацький інтерфейс для реєстрації нового користувача. Збирає дані (ім’я користувача, пароль, підтвердження пароля), відправляє запит на сервер та обробляє відповідь.

Головні методи.

__init__(self, main_window): конструктор, ініціалізує віджет та зберігає посилання на головне вікно.

init_ui(self): створює та повертає віджет з полями для введення даних реєстрації, кнопками “Register” та “Back”, а також міткою для попереджень.

on_register(self): асинхронний метод, який викликається при натисканні кнопки “Register”. Збирає введені дані, перевіряє їх на заповненість та відправляє запит на сервер через WebsocketClient.send_request з endpoint="register/". У разі успішної реєстрації викликає on_back_to_login(), інакше відображає помилку (рис. 3.8).

```
async def on_register(self):
    username = self.username_input.text()
    password = self.password_input.text()
    confirm_password = self.confirm_password_input.text()

    if not username or not password or not confirm_password:
        self.warning_label.setText("All fields are required!")
        # ... (відображення попередження) ...
        return

    response = await WebsocketClient.send_request(
        endpoint="register/",
        request_type="register",
        data={"username": username, "password": password, "confirm_password": confirm_password}
    )

    if response['status'] == 'success':
        self.on_back_to_login()
    else:
        self.warning_label.setText(response['message'])
        # ... (відображення попередження) ...
```

Рисунок 3.8 – Код методу відправки на сервер для перевірки введених користувачем даних для реєстрації класу RegisterWidget

on_back_to_login(self): викликається при натисканні кнопки “Back” або після успішної реєстрації. Створює екземпляр LoginWidget та встановлює його як центральний віджет головного вікна.

Клас AddFriendWidget(QFrame).

Призначення: надає спливаюче вікно (реалізоване як QFrame) для введення імені користувача, якого необхідно додати до списку друзів.

Головні методи.

__init__(self, main_window=None): конструктор, ініціалізує віджет, встановлює його як дочірній до головного вікна.

init_ui(self): створює інтерфейс віджета [7, 10]: поле для введення імені друга, кнопки “Add” та “Cancel” (реалізована як кнопка «←»), мітку для попереджень.

add_friend(self): асинхронний метод, який викликається при натисканні кнопки “Add”. Збирає ім’я друга, відправляє запит на сервер через WebsocketClient.send_request з endpoint='chat/' та request_type='add_friend'. У разі успіху оновлює список друзів в головному вікні (*self.main_window.load_friends()*) (рис. 3.9).

```
async def add_friend(self):
    friend_name = self.friend_input.text().strip()
    if not friend_name:
        self.warning_label.setText("Please enter a friend username!")
    else:
        response = await WebsocketClient.send_request(
            endpoint='chat/',
            request_type='add_friend',
            data={'username': self.main_window.username, 'friend_username': friend_name}
        )
        if response['status'] == 'unsuccess':
            self.warning_label.setText(f"{response['message']}")
        else:
            self.warning_label.setText("")
            await self.main_window.load_friends()
```

Рисунок 3.9 – Код методу відправки сигналу на сервер для додавання нового чату з користувачем класу AddFriendWidget

handle_close(self): викликається при натисканні кнопки “Cancel”. Закриває віджет AddFriendWidget та активує раніше деактивовані елементи головного вікна.

center_on_screen(self): центрує віджет відносно головного вікна.

Клас WebsocketClient.

Призначення: інкапсулює логіку роботи з WebSocket з'єднаннями [4] на стороні клієнта. Відповідає за встановлення з'єднань, відправку запитів на сервер, отримання та передачу повідомлень до MainWindow для обробки [9].

Головні методи.

connect_chat(self, chat_name): асинхронно встановлює WebSocket з'єднання [4] з сервером для конкретного чату [9] (рис. 3.10).

```
async def connect_chat(self, chat_name):
    self.chat_name = chat_name
    uri = f"{self.BASE_URI}chat/{chat_name}/"
    self.chat_websocket = await websockets.connect(uri)
    print(f"Connected to chat {chat_name}")
    asyncio.create_task(self.listen())
```

Рисунок 3.10 – Метод для встановлення з'єднання між клієнтом та сервером класу WebsocketClient

listen(self): асинхронно прослуховує вхідні повідомлення від сервера для активного чату та передає їх методу on_message_received об'єкта client (яким є MainWindow).

send_chat_message_or_file(self, is_file, sender, message=None, filename=None, filedata=None): асинхронно відправляє на сервер текстове повідомлення або дані файлу (рис. 3.11).

```
async def send_chat_message_or_file(self, is_file, sender, message=None, filename=None,
filedata=None):
    if not self.chat_websocket:
        print("Chat WebSocket is not connected")
        return
    data = None
    if is_file:
        data = {
            "request_type": "send_file", "sender": sender,
            "chat_name": self.chat_name, "filename": filename,
            "filedata": filedata
        }
    else: # Text message
        data = {
            "request_type": "send_message", "sender": sender,
            "chat_name": self.chat_name, "message": message
```

```
    }
await self.chat_websocket.send(json.dumps(data))
```

Рисунок 3.11 – Код методу-посередника відправки повідомлення або файлу користувача з клієнта на сервер класу WebsocketClient

connect_unread_counter(self, username): встановлює окреме WebSocket з'єднання [4] для отримання оновлень про кількість непрочитаних повідомлень [9].

listen_unread_counter(self, username): прослуховує повідомлення від сервера щодо оновлення лічильників непрочитаних повідомлень [9].

send_request(endpoint, request_type, data) (статичний метод): універсальний асинхронний метод для відправки одноразових запитів на сервер (наприклад, для логіну, реєстрації, додавання друга, завантаження файлу) та отримання відповіді [9] (рис. 3.12).

```
@staticmethod
async def send_request(endpoint, request_type, data):
    uri = f"{WebsocketClient.BASE_URI}{endpoint}"
    async with websockets.connect(uri) as websocket:
        request = {"request_type": request_type, **data}
        await websocket.send(json.dumps(request))
        response = await websocket.recv()
        print(f"Response received: {response}")
        return json.loads(response)
```

Рисунок 3.12 – Метод-посередник відправки сигналів з клієнта на сервер класу WebsocketClient

3.2.2 Серверна частина

Серверна частина, реалізована на Django [1] та Django Channels [2, 9], відповідає за обробку WebSocket з'єднань [4, 9], автентифікацію користувачів, керування логікою чатів, збереження історії повідомлень та файлів, а також взаємодію з базою даних для керування інформацією про користувачів та їхні списки друзів.

Клас ChatConsumer(AsyncWebsocketConsumer).

Призначення: головний конс'юмер для обробки WebSocket з'єднань [4], пов'язаних з чатами. Відповідає за прийом та відправку повідомлень, обробку файлів, завантаження історії чату, керування друзями та оновлення лічильників непрочитаних повідомлень [9].

Константи.

CHAT_HISTORY_DIR = "chat_history": директорія для збереження JSON-файлів з історією повідомлень.

CHAT_FILES_DIR = "chat_files": директорія для збереження завантажених файлів.

Головні методи.

connect(self): обробляє нове WebSocket з'єднання [4]. Якщо шлях вказує на конкретний чат, підписує конс'юмер на групу каналів цього чату та завантажує історію. Якщо шлях вказує на unread_counters, підписує на групи для оновлення лічильників непрочитаних повідомлень для друзів користувача [9].

disconnect(self, close_code): обробляє розрив WebSocket з'єднання [4], відписуючи конс'юмер від відповідних груп каналів [9].

receive(self, text_data): обробляє вхідні повідомлення від клієнта. Розбирає JSON-дані та на основі request_type викликає відповідний метод-обробник (наприклад, *handle_send_message*, *handle_get_user_friends*) (рис. 3.13).

```
async def receive(self, text_data):
    data = json.loads(text_data)
    request_type = data.get('request_type')
    if request_type == 'send_message':
        # ...
        await self.handle_send_message(sender, chat_name, message=message)
    elif request_type == 'send_file':
        # ...
        await self.handle_send_message(sender, chat_name, filename=filename, filedata=filedata)
    elif request_type == 'get_friends':
        # ...
        await self.handle_get_user_friends(username)
    # ... інші обробники ...
```

Рисунок 3.13 – Частковий код методу отримання та подальшої обробки сигналів від клієнта класу ChatConsumer

handle_send_message(self, sender, chat_name, message=None, filename=None, filedata=None): обробляє надсилання текстового повідомлення або файлу. Якщо це файл, він декодується з Base64, зберігається у CHAT_FILES_DIR, після чого інформація про повідомлення/файл зберігається в історії чату та розсилається всім учасникам чату через channel_layer.group_send. Також ініціює оновлення лічильника непрочитаних повідомлень.

chat_message(self, event): метод, який викликається Django Channels [2], коли надходить повідомлення для групи, на яку підписаний конс'юмер. Відправляє дані події (повідомлення) підключеному клієнту [9].

save_message_or_file_to_history(self, sender, timestamp, message=None, filename=None): зберігає інформацію про повідомлення або файл в JSON-файлі історії відповідного чату.

load_chat_history(self): завантажує історію повідомлень для поточного чату з JSON-файлу та відправляє її клієнту.

handle_get_user_friends(self, username): отримує список друзів для вказаного користувача з бази даних, разом з інформацією про фото профілю та кількість непрочитаних повідомлень для кожного друга, та відправляє ці дані клієнту.

handle_add_friend(self, username, friend_username): обробляє запит на додавання друга, перевіряючи існування користувачів та чи не є вони вже друзями, після чого додає зв'язок у базі даних.

update_unread_counter(self, chat_name, sender) та unread_message(self, event): відповідають за оновлення та розсилку інформації про кількість непрочитаних повідомлень відповідним користувачам.

download_file(self, chat_name, filename): обробляє запит на завантаження файлу, читає його з CHAT_FILES_DIR, кодує в Base64 та відправляє клієнту.

Клас LoginConsumer(AsyncWebsocketConsumer).

Призначення: обробляє WebSocket з’єднання [4] для автентифікації користувачів.

Головні методи.

receive(self, text_data): приймає JSON-дані з логіном та паролем від клієнта.

handle_login(self, username, password): асинхронно перевіряє облікові дані користувача, звертаючись до моделі CustomUser. Відправляє клієнту результат автентифікації (успіх/невдача) (рис. 3.14).

```
async def handle_login(self, username, password):
    user = await CustomUser.get_user(username=username)
    if user is None:
        await self.send(text_data=json.dumps({
            'status': 'unsuccess', 'message': 'Incorrect username'}))
    elif check_password(password, user.password):
        await self.send(text_data=json.dumps({
            'status': 'success', 'message': f'Login {username} successfully!'}))
    else:
        await self.send(text_data=json.dumps({
            'status': 'unsuccess', 'message': 'Incorrect password'}))
```

Рисунок 3.14 – Код методу обробки логіну користувача на сервері та відправка результату на клієнт класу LoginConsumer

Клас RegisterConsumer(AsyncWebsocketConsumer).

Призначення: обробляє WebSocket з’єднання для реєстрації нових користувачів.

Головні методи.

receive(self, text_data): приймає JSON-дані з логіном, паролем та підтвердженням пароля від клієнта.

Handle_register(self, username, password, confirm_password): асинхронно виконує валідацію даних (перевірка на існування користувача, співпадіння паролів) та, у разі успіху, створює нового користувача в базі даних за допомогою моделі CustomUser. Відправляє клієнту результат реєстрації (рис. 3.15).

```
async def handle_register(self, username, password, confirm_password):
    user = await CustomUser.get_user(username=username)
    if user is not None:
        await self.send(text_data=json.dumps({
            'status': 'unsuccess', 'message': 'The username is already taken!'}))
    elif password != confirm_password:
        await self.send(text_data=json.dumps({
            'status': 'unsuccess', 'message': 'Passwords do not match!'}))
    else:
        user = await CustomUser.create_user(username, password)
        if user is None: # Додано перевірку на успішне створення
            await self.send(text_data=json.dumps({
                'status': 'unsuccess', 'message': 'User creation error!'}))
        else:
            await self.send(text_data=json.dumps({
                'status': 'success', 'message': f'User {username} created successfully!'}))
```

Рисунок 3.15 – Код методу обробки реєстрації користувача на сервері і відправка результату на клієнт класу RegisterConsumer

Модель CustomUser(AbstractUser) (Django Model).

Призначення: представляє користувача системи в базі даних Django [1]. Ця модель успадковується від стандартної моделі django.contrib.auth.models.AbstractUser, що надає базові поля та функціональність для автентифікації (такі як username, password, email, is_staff, is_superuser тощо), та розширює її додатковими полями [5, 6].

Поля.

profile_photo = models.ImageField(upload_to='profile_photos/', null=True, blank=True): поле для зберігання фотографії профілю користувача. Файли завантажуватимуться до директорії profile_photos/ в межах MEDIA_ROOT. Поле може бути порожнім (null=True, blank=True).

friends = models.ManyToManyField('self', symmetrical=True, blank=True): поле типу «багато-до-багатьох» для зв'язку користувача з іншими користувачами (друзями). symmetrical=True означає, що якщо користувач А є другом користувача Б, то користувач Б автоматично є другом користувача А. Список друзів може бути порожнім (blank=True).

Методи.

__str__(self): повертає рядкове представлення об'єкта користувача, що включає ім'я користувача та його статус суперкористувача.

get_user(cls, username): класовий метод, декорований @sync_to_async для асинхронного виконання запиту до бази даних. Отримує користувача за його username. Повертає об’єкт CustomUser або None, якщо користувач не знайдений (рис. 3.16).

```
@classmethod
@sync_to_async
def get_user(cls, username):
    try:
        user = cls.objects.get(username=username)
        return user
    except cls.DoesNotExist:
        return None
```

Рисунок 3.16 – Код методу отримання користувача за його username моделі CustomUser

create_user(cls, username, password): класовий метод, декорований @sync_to_async. Створює нового користувача з вказаним username та password (пароль автоматично хешується). Повертає створений об’єкт CustomUser або None у разі помилки під час створення (рис. 3.17).

```
@classmethod
@sync_to_async
def create_user(cls, username, password):
    try:
        user = cls.objects.create_user(username=username, password=password)
        return user
    except Exception as e:
        print(f"Error creating user: {e}")
        return None
```

Рисунок 3.17 – Код методу створення нового користувача моделі CustomUser

Ця структура класів та їх взаємодія забезпечують функціональність сервісу для спілкування, включаючи реєстрацію, автентифікацію, обмін текстовими повідомленнями, обмін файлами, керування списком друзів та відображення статусу повідомлень.

3.3 Тестування

Тестування програмного забезпечення є невід’ємною частиною процесу розробки, метою якої є забезпечення якості, надійності та відповідності продукту заявленим вимогам. Для розробленого сервісу спілкування та обміну даними було проведено ручне функціональне тестування, що імітувало дії кінцевого користувача для перевірки основних сценаріїв взаємодії з системою.

3.3.1 Методологія тестування

Основний підхід до тестування – функціональне тестування методом «чорної скриньки». Це означає, що перевірка функціональності здійснювалася з точки зору кінцевого користувача, без урахування внутрішньої структури коду, шляхом взаємодії з графічним інтерфейсом клієнтського застосунку (PyQt6) [3] та спостереження за результатами (наприклад, відображення повідомлень, оновлення списків, збереження даних на сервері).

Тестування проводилося для наступних ключових компонентів та функцій:

– реєстрація та автентифікація користувачів;
– керування списком друзів;
– обмін текстовими повідомленнями в реальному часі;
– обмін файлами;
– відображення історії чату та статусу повідомлень.

3.3.2 Сценарії тестування та результати

Нижче наведено основні сценарії тестування працездатності, що були реалізовані з метою перевірки коректної роботи системи та її основного функціоналу.

Сценарій 1: реєстрація нового користувача.

Кроки:

1) запустити клієнтський застосунок;
2) на екрані входу натиснути кнопку “Register”;
3) у формі реєстрації ввести унікальне ім’я користувача, пароль та підтвердження пароля;
4) натиснути кнопку “Register”.

Очікуваний результат: система повинна успішно зареєструвати нового користувача, за умови, що введені дані є коректними та ім’я користувача не використовується іншими обліковими записами. Після успішної реєстрації користувач автоматично повертається до екрана входу.

Фактичний результат: сценарій виконано успішно. Новий користувач був створений, і система перейшла до екрану входу.

На рисунку 3.18 представлено інтерфейс реєстрації користувача.

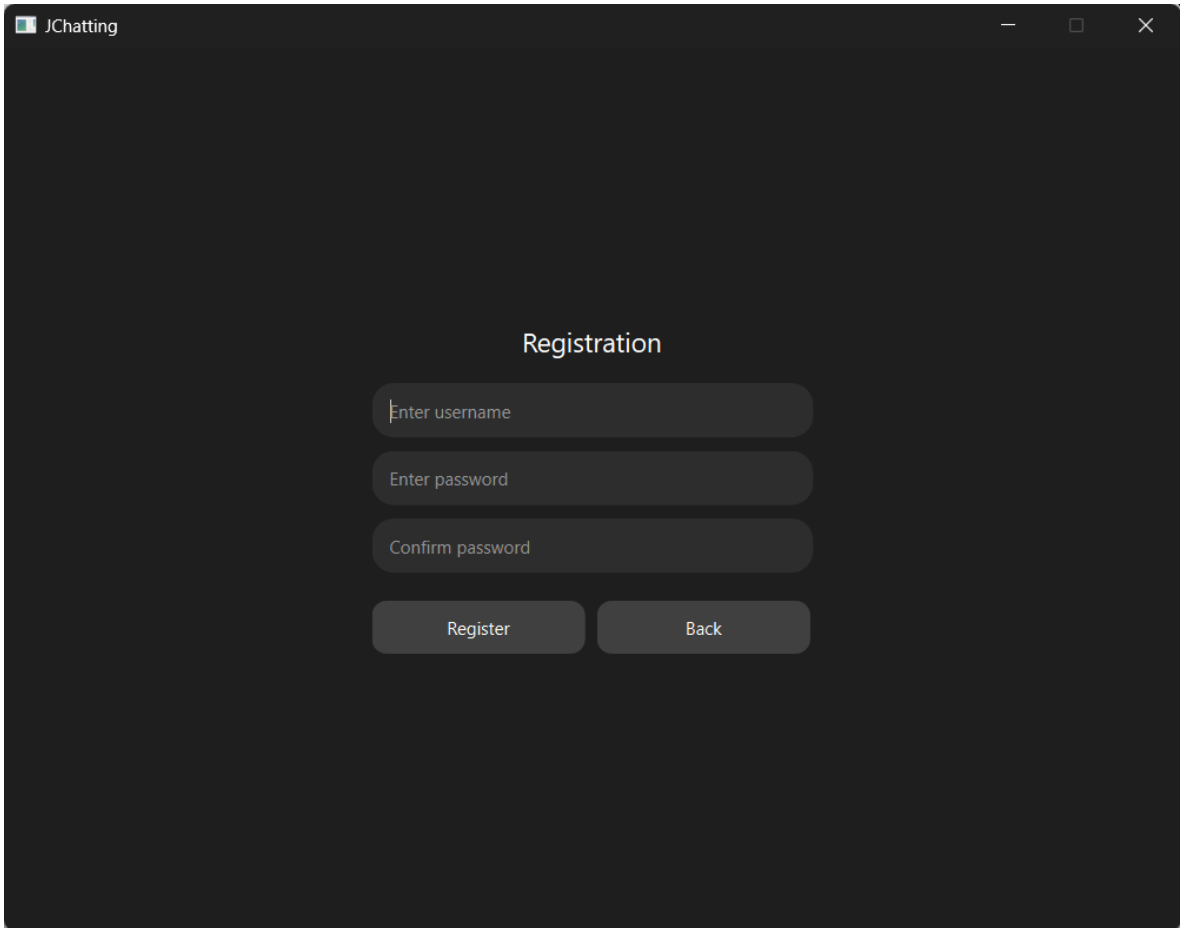


Рисунок 3.18 – Вигляд інтерфейсу реєстрації

Сценарій 2: помилка реєстрації (невідповідність паролів).

Кроки:

1) запустити клієнтський застосунок;
2) на екрані входу натиснути кнопку “Register”;
3) у формі реєстрації ввести унікальне ім’я користувача, пароль та інший пароль у полі підтвердження;
4) натиснути кнопку “Register”.

Очікуваний результат: система відображає повідомлення про помилку, що паролі не співпадають. Реєстрація не відбувається.

Фактичний результат: сценарій виконано успішно. Система відобразила відповідне повідомлення про помилку.

На рисунку 3.19 показано повідомлення про помилку при невідповідності паролів.

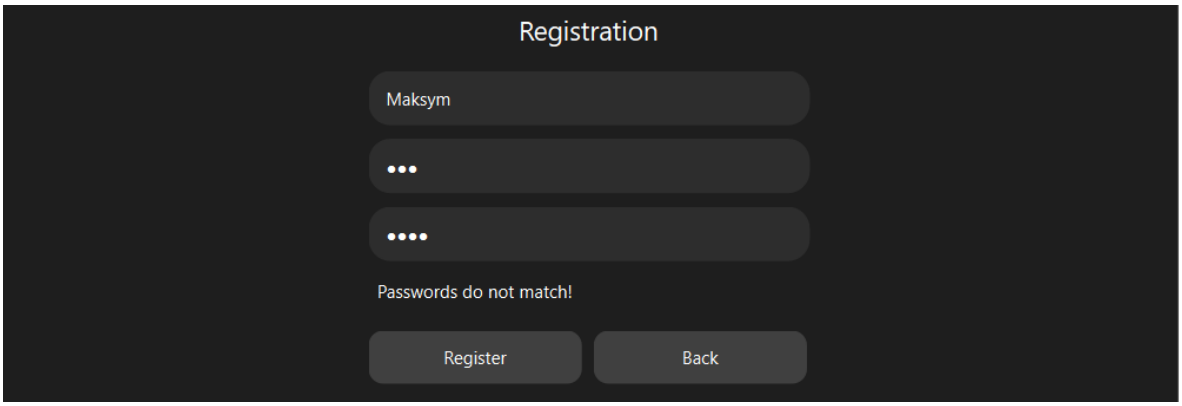


Рисунок 3.19 – Помилка невідповідності паролів при реєстрації облікового запису

Сценарій 3: авторизація існуючого користувача.

Кроки:

1) запустити клієнтський застосунок;
2) на екрані входу ввести коректні ім’я користувача та пароль;
3) натиснути кнопку “Login”.

Очікуваний результат: система успішно автентифікує користувача та відображає головний інтерфейс чату зі списком друзів.

Фактичний результат: сценарій виконано успішно. Користувач був автентифікований, відобразився головний екран застосунку (рис. 3.20 – 3.21).

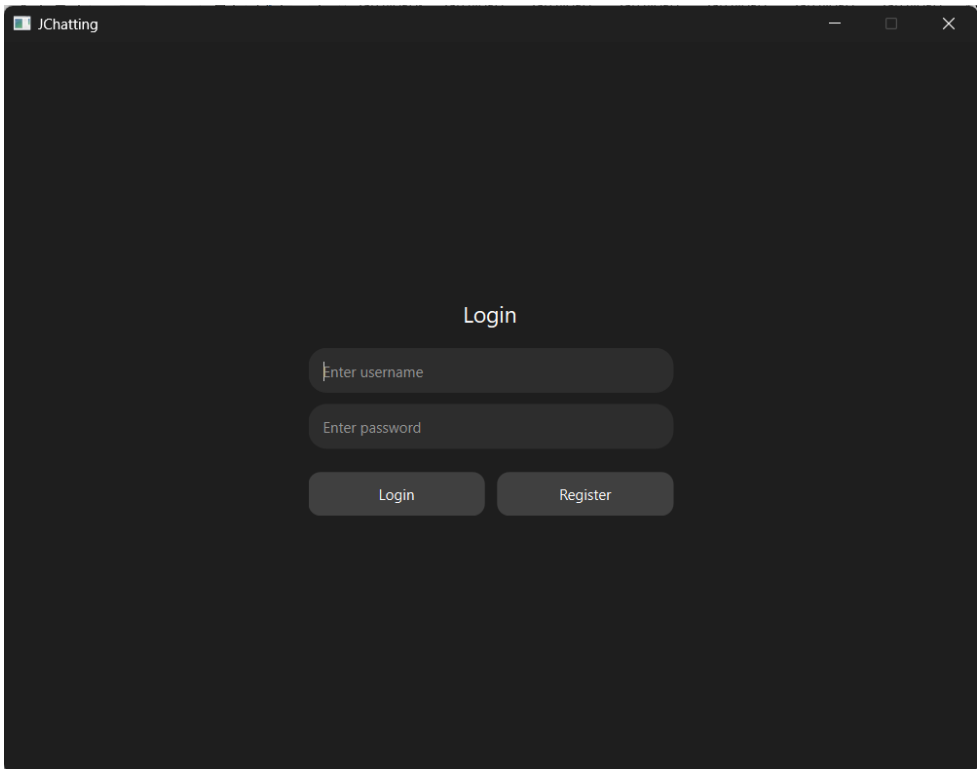


Рисунок 3.20 – Вигляд екрану входу в обліковий запис

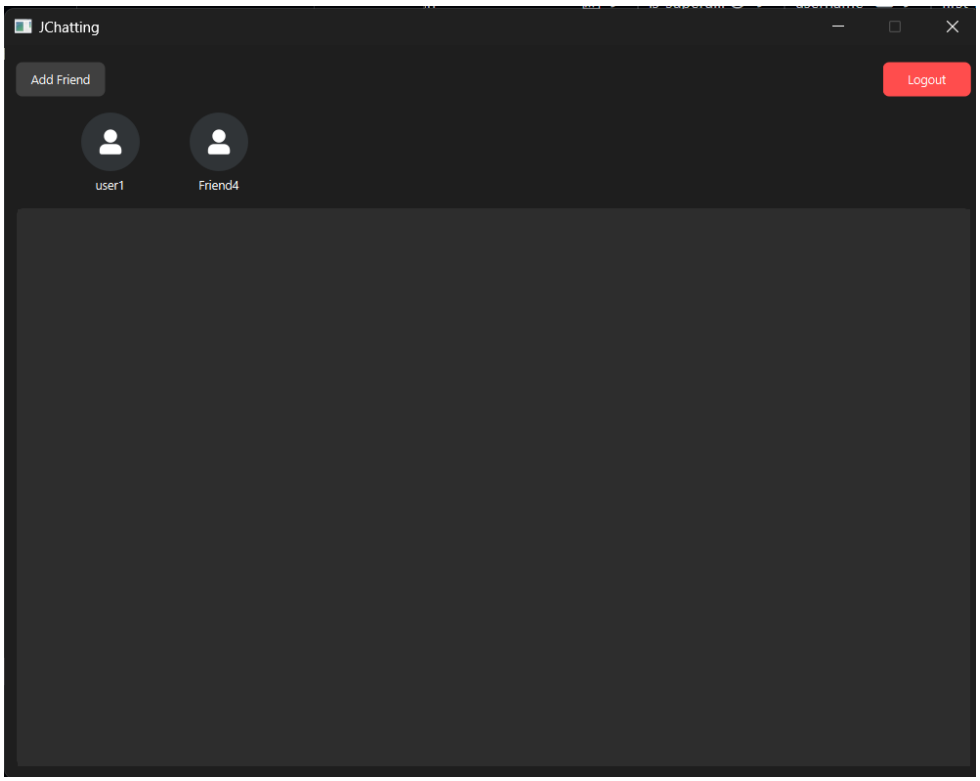


Рисунок 3.21 – Вигляд головного екрану застосунку після успішного логіну

Сценарій 4: додавання нового друга.

Кроки:

1) авторизуватися в системі;
2) натиснути кнопку “Add Friend”;
3) у вікні, що з’явилося, ввести ім’я існуючого в системі користувача (який ще не є другом);
4) натиснути кнопку “Add”.

Очікуваний результат: система додає вказаного користувача до списку друзів. Список друзів на головному екрані оновлюється, відображаючи нового друга.

Фактичний результат: сценарій виконано успішно. Новий друг був доданий, і список друзів оновився (рис. 3.22 – 3.23).

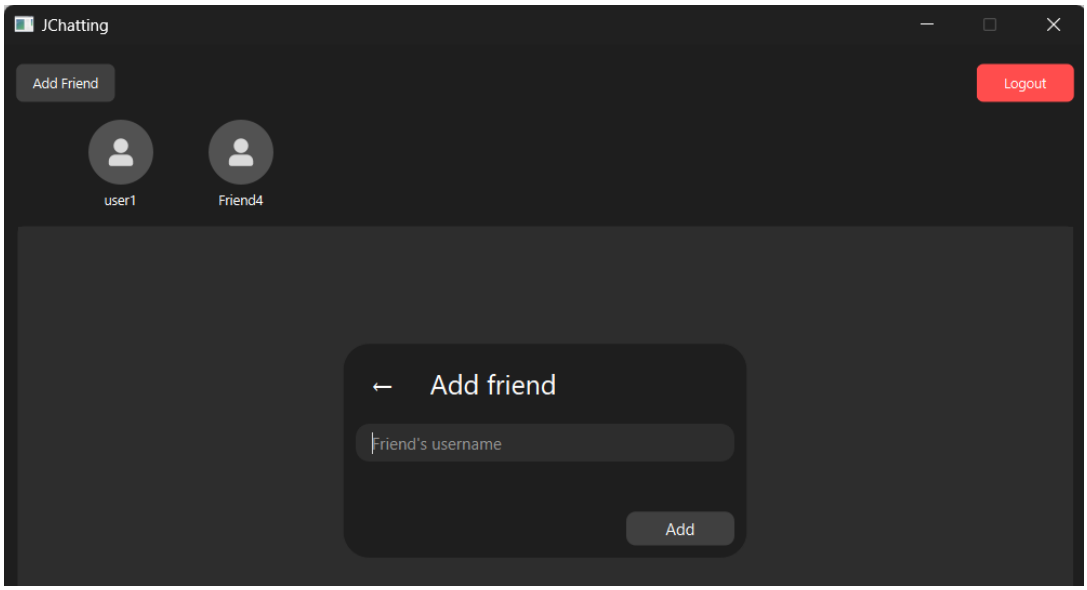


Рисунок 3.22 – Вигляд віджету додавання нового чату з користувачем

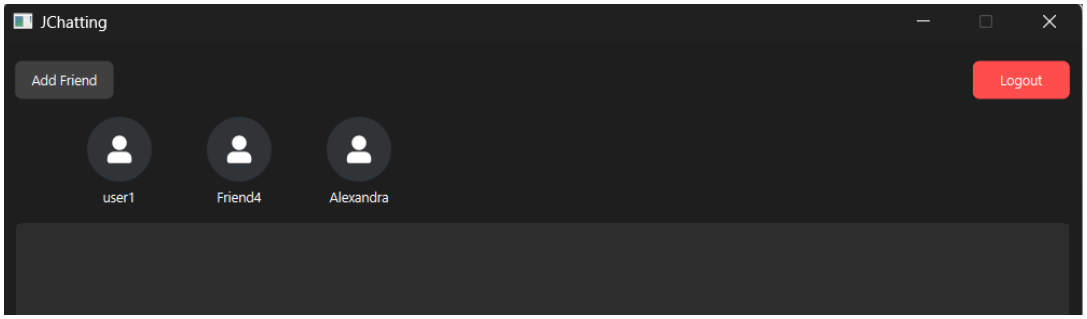


Рисунок 3.23 – Вигляд списку друзів після додавання друга “Alexandra”

Сценарій 5: відкриття чату з другом та надсилання текстового повідомлення.

Кроки:

1) авторизуватися в системі (Користувач 1);
2) авторизуватися в системі на іншому клієнті або під іншим обліковим записом (Користувач 2), який є другом Користувача 1;
3) користувач 1 обирає Користувача 2 зі списку друзів;
4) в активному вікні чату Користувач 1 вводить текстове повідомлення та натискає “Send”.

Очікуваний результат: повідомлення Користувача 1 відображається у його вікні чату. Користувач 2 отримує це повідомлення в реальному часі у своєму вікні чату з Користувачем 1. Лічильник непрочитаних повідомлень у Користувача 2 (якщо чат не був активним) оновлюється.

Фактичний результат: сценарій виконано успішно. Повідомлення було доставлено та відображено коректно в обох користувачів (рис. 3.24).

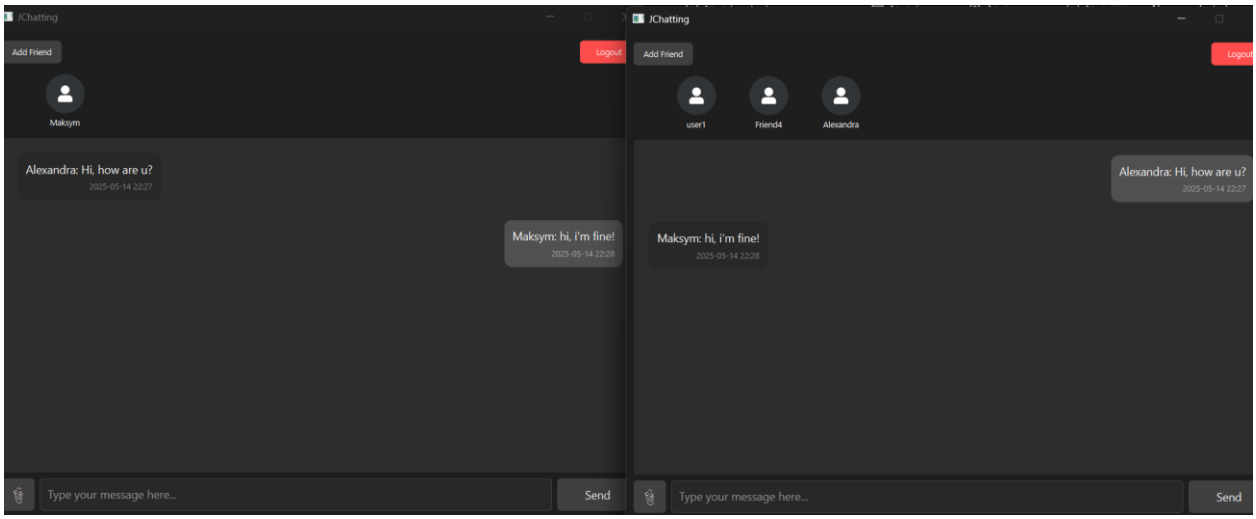


Рисунок 3.24 – Відображення повідомлення в обох чатах користувачів

Сценарій 6: надсилання файлу.

Кроки:

1) в активному чаті (як у Сценарії 5) Користувач 1 натискає кнопку для прикріплення файлу (іконка скріпки);

2) обирає файл зі свого комп’ютера (розмір файлу в межах допустимого).

Очікуваний результат: інформація про відправлений файл (ім’я файлу) відображається у вікні чату Користувача 1. Користувач 2 отримує інформацію про файл у своєму вікні чату та може його завантажити.

Фактичний результат: сценарій виконано успішно. Файл був відправлений, інформація про нього відобразилася, і інший користувач зміг його побачити (рис. 3.25).

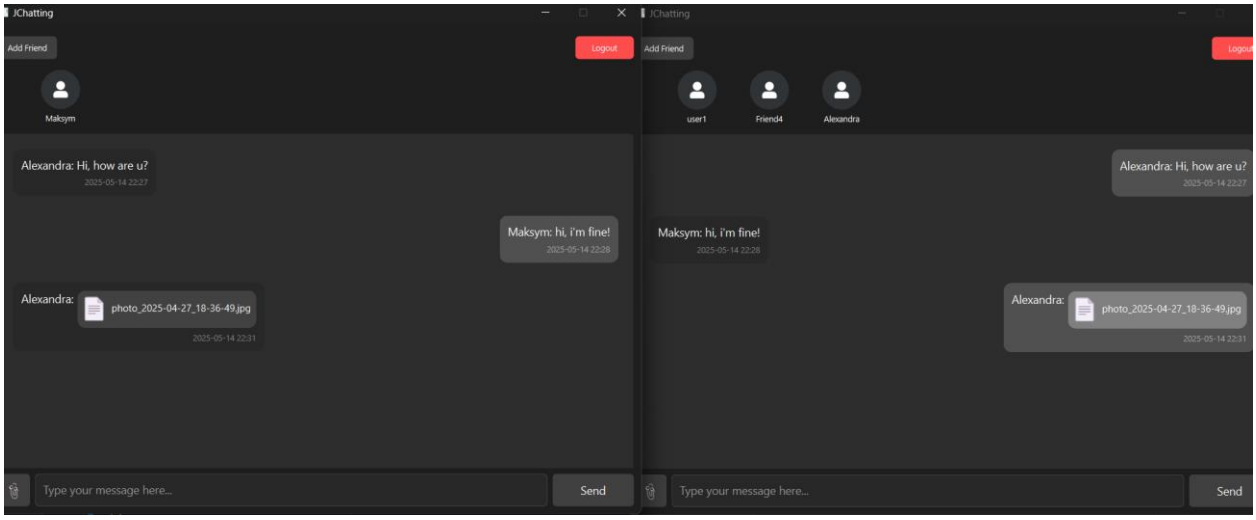


Рисунок 3.25 – Відображення відправленого файлу в обох чатах користувачів

Сценарій 7: завантаження отриманого файлу.

Кроки:

1) користувач 2 (який отримав файл у Сценарії 6) натискає на повідомлення з файлом у вікні чату;
2) обирає місце для збереження файлу.

Очікуваний результат: файл успішно завантажується на комп’ютер Користувача 2.

Фактичний результат: сценарій виконано успішно. Файл був завантажений у вказане місце (рис. 3.26).

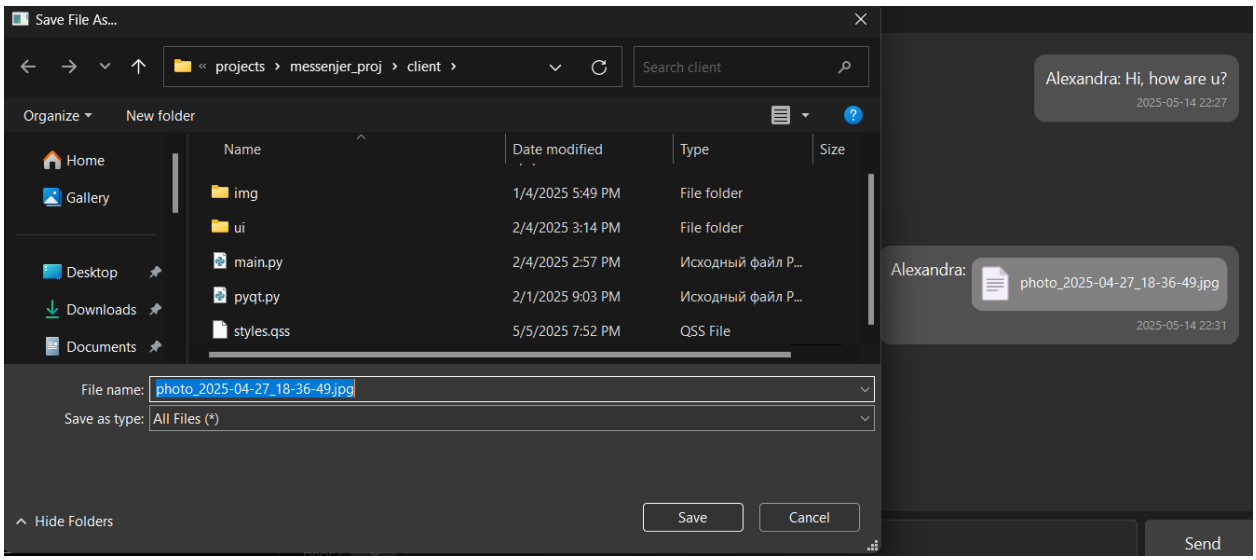


Рисунок 3.26 – Завантаження файла з чату

Сценарій 8: відображення лічильника непрочитаних повідомлень.

Кроки:

1) користувач 1 надсилає повідомлення Користувачу 2;
2) користувач 2 не має активного чату з Користувачем 1 (наприклад, відкритий чат з іншим другом або просто головний екран).

Очікуваний результат: біля іконки/імені Користувача 1 у списку друзів Користувача 2 з’являється лічильник непрочитаних повідомлень.

Фактичний результат: сценарій виконано успішно. Лічильник непрочитаних повідомлень коректно відобразився (рис. 3.27).



Рисунок 3.27 – Відображення лічильника непрочитаних повідомлень від користувача

Сценарій 9: вихід із системи (Logout).

Кроки:

1) авторизований користувач натискає кнопку “Logout”.

Очікуваний результат: система завершує сеанс користувача та повертає його на екран входу. WebSocket з’єднання [4] розриваються.

Фактичний результат: сценарій виконано успішно. Користувач був перенаправлений на екран входу.

Всі перелічені вище сценарії були успішно виконані. Але це далеко не всі сценарії тестування, які було проведено для розроблених виключних ситуацій. Ось перелік того, що ще було реалізовано:

- введено некоректне ім’я користувача або пароль при вході;
- спроба зареєструвати акаунт з вже існуючим ім’ям користувача;
- спроба додати не існуючого користувача у список чатів;
- успішне додавання користувача в БД після його реєстрації у застосунку;
- спроба додати у список чатів користувача, який вже там є;
- сценарій прокрутки чата до першого непрочитаного повідомлення при відкриті;
- спроба додати в список чатів самого себе;
- неможливість відправки порожнього повідомлення;
- перевірка успішного збереження повідомлень в історії чата;
- перевірка успішного збереження файлів на сервері;
- успішне оновлення лічильника непрочитаних повідомлень, після того, як користувач прочитав повідомлення.

Під час тестування функціонал системи працював коректно, відповідно до очікувань. Виявлені незначні недоліки в інтерфейсі або логіці виправлялися на етапі розробки, що дозволило досягти стабільної роботи застосунку.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було розроблено сервіс для спілкування та обміну даними, що використовує сучасні технології веб та десктопної розробки, а саме фреймворк Django [1] для серверної частини, бібліотеку PyQt6 [3] для створення клієнтського застосунку та протокол WebSocket [4] для забезпечення обміну повідомленнями в реальному часі.

Опишемо основні досягнення та результати роботи.

Проведено аналіз предметної області та існуючих рішень. Було досліджено ключові вимоги до систем обміну повідомленнями, проаналізовано переваги та недоліки популярних месенджерів, що дозволило сформувати чітке уявлення про необхідний функціонал та архітектуру розроблюваного сервісу.

Обґрунтовано вибір технологічного стеку. Django обрано завдяки швидкій розробці, підтримці баз даних, автентифікації та WebSocket через Django Channels. PyQt6 використано за гнучкість у створенні кросплатформних інтерфейсів і підтримку асинхронної взаємодії.

Розроблено архітектуру системи. Система побудована за клієнт-серверною архітектурою. Серверна частина на Django відповідає за бізнес-логіку, автентифікацію, керування чатами, збереження історії повідомлень та файлів. Клієнтська частина на PyQt6 забезпечує користувацький інтерфейс для взаємодії з сервісом. Комунікація між клієнтом та сервером відбувається через WebSocket, що дозволяє миттєво доставляти повідомлення.

Реалізовано ключовий функціонал сервісу:

- **реєстрація та автентифікація користувачів**: забезпечено безпечний вхід до системи та створення нових облікових записів;
- **керування списком друзів**: користувачі мають можливість додавати інших користувачів до свого списку друзів;
- **відображення історії чату**: забезпечено завантаження та відображення попередніх повідомлень при відкритті чату;

- **обмін текстовими повідомленнями**: реалізовано можливість надсилання та отримання текстових повідомлень у приватних чатах в режимі реального часу;
- **обмін файлами**: користувачі можуть обмінюватися файлами, які зберігаються на сервері та доступні для завантаження учасникам чату;
- **сповіщення про непрочитані повідомлення**: реалізовано візуальне сповіщення про наявність нових непрочитаних повідомлень від друзів.

Застосовано асинхронне програмування. Використання asyncio на клієнтській стороні та можливостей Django Channels на сервері забезпечило ефективну обробку мережевих операцій і взаємодію з інтерфейсом без блокування основного потоку.

Розроблено користувацький інтерфейс. Створено інтуїтивно зрозумілий графічний інтерфейс для клієнтського застосунку, що забезпечує зручну взаємодію користувача з усіма функціями сервісу.

Проведено тестування системи. Було виконано ручне функціональне тестування основних сценаріїв використання, що підтвердило працездатність розробленого функціоналу та відповідність поставленим вимогам.

Практична значущість роботи полягає у створенні функціонального прототипу сервісу для спілкування, який демонструє можливість інтеграції технологій Django, PyQt6 та WebSocket для розробки сучасних інтерактивних застосунків. Отриманий досвід та розроблені компоненти можуть бути використані як основа для подальшого розвитку та створення більш складних систем обміну інформацією.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна документація Django. *Django Project*. URL: https://docs.djangoproject.com (дата звернення 11.02.2025).

2. Офіційна документація Django Channels. *Django Channels – Channels 4.2.0 documentation*. URL: https://channels.readthedocs.io (дата звернення: 11.02.2025).

3. PyQt6 Reference Guide. *Reference Guide – PyQt Documentation v6.9.0*. URL: https://www.riverbankcomputing.com/static/Docs/PyQt6/ (дата звернення: 15.03.2025).

4. WebSockets API. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 15.03.2025).

5. Greenfeld D. R., Greenfeld A. R. Two Scoops of Django 3.x: Best Practices for the Django Framework. San Diego : Two Scoops Press, 2020. P. 207–229.

6. Vincent W. S. Django for Professionals: Production-ready Django development. San Francisco, 2021. P. 175–184.

7. Fitzpatrick M. Create GUI Applications with Python & Qt6. URL: https://www.pythonguis.com/pyqt6-book/ (дата звернення: 15.03.2025).

8. Ramalho L. Fluent Python: Clear, Concise, and Effective Programming : 2nd ed. Sebastopol : O'Reilly Media, 2022. P. 812–826.

9. Working with WebSockets in Django with Channels. *Real Python*. URL: https://realpython.com/getting-started-with-django-channels/ (дата звернення: 14.04.2025).

10. Python PyQt6 – learn GUI programming with Python and PyQt6. *ZetCode*. URL: https://zetcode.com/pyqt6/ (дата звернення: 15.03.2025).