Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЕРЕДАЧІ ПОВІДОМЛЕНЬ У РЕАЛЬНОМУ ЧАСІ
SOFTWARE FOR REAL TIME MESSAGE TRANSMISSION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-113сп
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення
Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ЛИТВИН Д.К.
(ПРІЗВИЩЕ та ініціали)
Керівник ОЛІЙНИК А.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент КОЦУР М.І.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
“____” _____2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЛИТВИНА Дмитра Костянтиновича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для передачі повідомлень у реальному часі. Software for Real Time Message Transmission

керівник проєкту (роботи) д.т.н., професор, ОЛІЙНИК Андрій Олександрович ,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 7 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 03 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ОЛІЙНИК А.О., професор		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 7 ” квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка) _____ Дмитро ЛИТВИН
(підпис) (Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Андрій ОЛІЙНИК
(підпис) (Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 94 с., 3 табл., 29 рис., 3 дод., 19 джерел.

ECLIPSE, JAVA, МОВА ПРОГРАМУВАННЯ, ПОВІДОМЛЕННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СЕРЕДОВИЩЕ РОЗРОБКИ.

Об’єкт дослідження – процес розробки програмного забезпечення для передачі повідомлень у реальному часі.

Предмет дослідження – програмні засоби для передачі повідомлень у реальному часі.

Мета роботи – розробка програмного забезпечення для передачі повідомлень у реальному часі для підвищення швидкості обміну даними між користувачами.

Матеріали, методи та технічні засоби: мова програмування Java, середовище розробки Eclipse.

Результати. Розроблено проєктні рішення для створення програмного забезпечення для передачі повідомлень у реальному часі. Створено програмне забезпечення для передачі повідомлень у реальному часі. Проведено тестування розробленої програмної системи для передачі повідомлень у реальному часі.

Висновки. Мету роботи досягнуто. Розроблено програмне забезпечення для передачі повідомлень у реальному часі за допомогою мови програмування Java та середовища розробки Eclipse.

Галузь використання – корпоративні застосунки.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 94 pages, 3 tables, 29 figures, 3 appendixes, 19 sources.

ECLIPSE, JAVA, PROGRAMMING LANGUAGE, MESSAGE, SOFTWARE, DEVELOPMENT ENVIRONMENT.

The object of research is the process of developing software for real-time messaging.

The subject of the research is software for real-time messaging.

The purpose of this work is to develop software for real-time messaging to increase the speed of data exchange between users.

Materials, methods and technical tools: Java programming language, Eclipse development environment.

Results. Project solutions for the creation of software for real-time messaging has been designed. The software to support the process of real-time messaging has been developed. The developed software system for real-time messaging has been tested.

Conclusions. The goal of the work has been achieved. The software tools for real-time messaging using the Java programming language and the Eclipse development environment have been developed.

Scope of use – corporate applications.

ЗМІСТ

С.

Перелік скорочень та умовних познак 8
Вступ 9
1 Аналіз предметної області 11
1.1 Програмне забезпечення для передачі повідомлень у реальному часі ... 11
1.2 Аналіз програмного забезпечення для передачі повідомлень у реальному часі 21
1.3 Висновки за розділом 1 31
2 Матеріали і методи 32
2.1 Вибір мови програмування 32
2.2 Вибір середовища розробки для створення програмного забезпечення для передачі повідомлень у реальному часі 36
2.3 Висновки за розділом 2 40
3 Опис програми 41
3.1 Структура програмного забезпечення для передачі повідомлень у реальному часі 41
3.2 Функціонування програмного забезпечення для передачі повідомлень у реальному часі 43
3.3 Розробка бази даних програмного забезпечення для передачі повідомлень у реальному часі 46
3.4 Проєктування інтерфейсу програмного забезпечення для передачі повідомлень у реальному часі 48
3.5 Висновки за розділом 3 50
4 Експлуатація, тестування та експериментальне дослідження програми 52
4.1 Призначення й умови застосування програми 52
4.2 Характеристики програми для передачі повідомлень у реальному часі 53
4.3 Інструкція по експлуатації програми 54
4.3.1 Звернення до програми 54
4.3.2 Вхідні й вихідні дані 55

4.3.3 Повідомлення 55
4.4 Виконання програмного забезпечення для передачі повідомлень у реальному часі 55
4.5 Тестування програмного забезпечення для передачі повідомлень у реальному часі 58
4.6 Висновки за розділом 4 59
Висновки 60
Перелік джерел посилання 63
Додаток А Технічне завдання 65
Додаток Б Фрагмент тексту програми 70
Додаток В Слайди презентації 88

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

IDE – Integrated Development Environment;

JRE – Java Runtime Environment;

JDK – Java Development Kit;

JSON – Java Script Object Notation;

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Актуальність програмного забезпечення для передачі повідомлень у реальному часі зумовлюється зростанням потреби в миттєвій комунікації, яка перестала бути допоміжною функцією й перетворилася на один з ключових механізмів взаємодії між людьми, організаціями та технічними системами. У сучасному цифровому середовищі повідомлення перестають бути простою формою обміну даними, а набувають статусу основного каналу координації, контролю та прийняття рішень. Сам момент їх доставки важливий настільки ж, як і сам зміст, оскільки від швидкості обміну часто залежить якість процесів, безпека операцій та ефективність роботи [1], [2].

Потреба в таких програмних системах загострюється розвитком мобільних технологій, коли комунікація супроводжує людину впродовж усього дня, і будь-яка затримка сприймається як порушення звичного ритму діяльності. У корпоративному середовищі миттєвий обмін повідомленнями перетворюється на ядро цифрової організації, де координація задач, підтримка клієнтів і внутрішня взаємодія будуються навколо інструментів швидкої реакції. В умовах глобальної розподіленості команд саме реальний час дає можливість працювати так, ніби всі учасники перебувають в одному просторі, знімаючи бар’єри фізичної відстані та часових зон [3], [4].

Особливу вагу таким програмним рішенням надає зростання кількості сервісів, що працюють безперервно. У фінансових системах оперативність визначає здатність своєчасно реагувати на транзакції та ризики. У сфері електронної комерції миттєва комунікація дозволяє супроводжувати клієнта в момент покупки, забезпечуючи швидку підтримку та підвищуючи довіру. У промислових та інфраструктурних комплексах повідомлення реального часу слугують основою моніторингу та управління, забезпечуючи негайне виявлення відхилень та швидку реакцію на критичні події [5], [6].

Проте деякі програми для передачі повідомлень у реальному часі можуть не забезпечувати стабільну роботу в умовах високого навантаження. У

таких програмних системах навіть незначні збої можуть призвести до втрати важливих повідомлень або затримки їх доставки, що одразу впливає на користувацький досвід і порушує робочі процеси. Залежність від швидкості мережі створює додаткові ризики, оскільки будь-які коливання пропускної здатності чи нестабільність з'єднання миттєво позначаються на якості передавання даних. Проблемою залишається і забезпечення безпеки. Передавання даних у реальному часі потребує постійного шифрування та захисту каналів зв'язку, а також контролю за цілісністю даних у момент їх передачі. Будь-які помилки у конфігурації або недосконаліть механізмів аутентифікації створюють потенційні точки доступу для зловмисників. Поєднання високої швидкості роботи та необхідності негайної реакції іноді зменшує можливість застосування надто громіздких механізмів захисту, що збільшує ризики витоку інформації. У багатьох випадках користувачі змушені довіряти стороннім провайдерам критично важливу інформацію, що не завжди відповідає їхнім вимогам до приватності [1]-[6].

Тому актуальною є розробка програмного забезпечення для передачі повідомлень у реальному часі. У дипломній кваліфікаційній роботі бакалавра розв'язується актуальне завдання розробки програмного забезпечення для передачі повідомлень у реальному часі.

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

– виконати аналіз предметної області та програмних засобів для передачі повідомлень у реальному часі;

– здійснити проєктування програмного забезпечення для передачі повідомлень у реальному часі;

– створити програмне забезпечення для передачі повідомлень у реальному часі;

– виконати тестування розробленого програмного забезпечення для передачі повідомлень у реальному часі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Програмне забезпечення для передачі повідомлень у реальному часі

Обмін інформацією, що передається без затримок, передбачає миттєве надходження та відображення даних між співрозмовниками, незалежно від їх кількості. Платформи, які використовують такий підхід, здатні передавати значні обсяги повідомлень у реальному часі, забезпечуючи постійний доступ до оновленого змісту. Такий спосіб спілкування створений для того, щоб користувач отримував відповіді одразу після їх надсилання. У сучасних умовах застосовуються як форми взаємодії, що вимагають одночасної присутності учасників, так і такі, що дозволяють продовжувати обговорення незалежно від того, чи перебувають співрозмовники онлайн [1].

На відміну від цього, традиційна передача повідомлень може супроводжуватися помітними затримками. Наприклад, електронна кореспонденція не гарантує швидкого перегляду, оскільки адресат може ознайомитися з нею значно пізніше. Проте кожен із підходів має свою роль, оскільки люди можуть віддавати перевагу різним каналам спілкування. У середовищі, де цінуються швидкість реакції та постійний контакт, важливо підтримувати безперервну взаємодію з аудиторією [1].

Обмін повідомленнями без відкладень охоплює будь-який формат діалогу, у якому інформація передається миттєво. Такий спосіб характерний для онлайн чатів і популярних месенджерів. Система дозволяє усунути часові перерви між зверненням і відповіддю, що особливо корисно для тих, хто знайомиться з Інтернет-ресурсом і хоче швидко отримати уточнення. Швидка реакція підвищує шанс того, що людина продовжить взаємодію з ресурсом або здійснить корисну дію [1].

Миттєвість передачі може забезпечуватися технологіями push-механізмів, коли кожне нове повідомлення негайно надходить іншим учасникам через платформу, що постійно оновлює стан діалогу. За умови

стабільного підключення інформація відображається одразу, і користувачі можуть продовжувати обговорення так, ніби перебувають у реальній розмові. У повсякденному житті подібна форма комунікації використовується постійно. Вона охоплює роботу чат-ботів, спілкування з друзями чи колегами, а також підтримку взаємодії у професійних інструментах для командної роботи. Такий підхід дає бізнесу можливість підвищувати ефективність внутрішньої взаємодії та покращувати взаємини з клієнтами. Миттєві повідомлення виступають складовою цієї системи, дозволяючи швидко звертатися до потрібних людей і передавати необхідну інформацію [1].

У корпоративному середовищі цей тип обміну дає змогу підтримувати контакт між різними підрозділами та зовнішніми співробітниками. Інструменти, створені для спільної роботи, забезпечують як одночасне спілкування, так і можливість надсилання інформації без потреби перебувати в мережі в один момент. Подібне рішення особливо важливе для груп, яким потрібно залишатися на зв’язку в будь-який момент [1], [2].

Найбільшою перевагою миттєвої комунікації є повна незалежність від місцезнаходження. Дані передаються однаково швидко як через мобільні мережі, так і через бездротове підключення, що дозволяє підтримувати спілкування в умовах, коли фізична відстань не має значення [1], [2].

Важлива роль цієї технології помітна також у сфері підтримки клієнтів. Якщо людині потрібно уточнити деталі про товар або послугу, можливість отримати відповідь без очікування значно покращує її досвід взаємодії з компанією. Використання живого чату разом із автоматизованими системами забезпечує доступність допомоги у будь-який момент, а також дозволяє швидко опрацьовувати типові запити. Це підвищує рівень сервісу та підтримує зацікавленість потенційних покупців [1], [2].

Оперативний обмін повідомленнями покращує співпрацю з клієнтами та сприяє тому, що запити, які могли бути втрачені, перетворюються на реальні угоди. Крім того, можливість миттєво отримувати інформацію сприяє підвищенню ефективності роботи. Користувачі отримують швидкий доступ до

зворотного зв’язку, аналітичних даних і реакцій аудиторії, що дозволяє своєчасно ухвалювати рішення в умовах стрімкого ринку [1], [2].

Програмні засоби для передачі повідомлень у реальному часі забезпечують миттєву взаємодію між користувачами та дозволяють інформації надходити без затримок. Таке програмне забезпечення здатне обробляти повідомлення відразу після їхнього створення, що робить спілкування максимально природним і наближеним до живого діалогу. Подібні програмні рішення створюються для того, щоб підтримувати постійний потік даних між учасниками комунікації, незалежно від їхнього місцезнаходження чи типу пристрою [2], [3].

Головна ідея полягає в тому, що користувач отримує доступ до нових даних у ту ж мить, коли вони з’являються в іншого учасника взаємодії. Такий механізм реалізується через технології, що підтримують безперервне з’єднання між клієнтом і сервером, забезпечуючи постійне оновлення інформації. Завдяки цьому можливо досягати ефекту одночасної присутності, який робить обговорення гнучким, динамічним та безперервним. Системи цього типу здатні витримувати значні навантаження, оскільки кількість повідомлень, що циркулюють через них, може зростати до дуже великих обсягів [2], [3].

Особливістю такого програмного забезпечення є орієнтація на миттєву реакцію. Дані передаються не в межах фіксованих інтервалів, а відправляються одразу після їх створення, тому користувачам не потрібно чекати синхронізації. Це забезпечує максимально плавний перебіг розмови та дозволяє приймати рішення, спираючись на найактуальнішу інформацію. Завдяки цьому формується середовище, у якому ефективне співробітництво стає простішим, а час виконання завдань скорочується [2], [3].

Суттєвою перевагою таких програмних систем є можливість організовувати взаємодію без обмежень, пов’язаних із відстанню. Люди можуть підтримувати зв’язок у будь-який момент, якщо мають доступ до Інтернету. Це особливо важливо в контексті роботи команд, що розташовані в

різних регіонах, а також для бізнесів, які прагнуть оперативно реагувати на звернення своїх клієнтів. Швидке реагування зменшує ризик втрати зацікавленості та формує довіру до сервісу чи компанії [2], [3].

Програмні платформи такого типу дозволяють поєднувати обмін текстовими повідомленнями, передавання мультимедійних даних та інші форми цифрової взаємодії в одному середовищі. Гнучка структура дозволяє адаптувати функціональність під потреби різних сфер: від приватного спілкування до складних корпоративних комунікацій. Вони також можуть включати механізми асинхронної роботи, коли повідомлення зберігаються та доставляються відразу після появи з'єднання, що забезпечує безперервність діалогу за будь-яких умов [2], [3].

Підвищення ефективності є одним із ключових результатів використання такого підходу. Миттєвий доступ до інформації дає змогу швидше виявляти потреби, приймати рішення та координувати дії. У середовищі, де швидкість реакції має вирішальне значення, подібне програмне забезпечення допомагає уникати затримок, які зазвичай виникають у традиційних формах обміну даними [2], [3].

Загалом, програмне забезпечення для передачі повідомлень у реальному часі формує інструментальну основу для сучасної комунікації. Воно дозволяє будувати взаємодію таким чином, щоб вона відбувалася без перерв і з урахуванням постійної доступності даних. Завдяки цьому забезпечується висока якість зв'язку, оперативне прийняття рішень і комфортне спілкування, що робить такі технології невід'ємною частиною цифрового середовища [2], [3].

Обмін повідомленнями в реальному часі трактується як технологія, що забезпечує миттєве передавання даних між двома чи більше сторонами без помітних затримок. У такому підході використовуються механізми, які підтримують безперервний зв'язок між учасниками передавання, завдяки чому реакція на події відбувається практично одночасно з їх виникненням. API, створені для цього призначення, дають змогу програмам або пристроям

взаємодіяти за допомогою каналів, де дані надходять одразу після надсилання, забезпечуючи стабільний низьколатентний потік інформації у двох напрямках. Зазвичай такі інструменти функціонують на основі WebSockets, MQTT або інших технологій тривалих з’єднань, що дає можливість організувати швидкий та керований рух даних у найрізноманітніших програмних середовищах [3], [4].

У процесі передавання інформації в режимі реального часу використовується підхід, який дає можливість розподіляти та приймати повідомлення у вигляді невеликих блоків із постійною періодичністю. Задля цього застосовуються гнучкі комунікаційні схеми, де підтримується публікація даних від одного джерела та автоматичне отримання їх усіма зацікавленими сторонами, що приєднані до відповідного каналу. Самі дані переважно структуруються за допомогою універсальних форматів передавання даних на кшталт JSON, що дає змогу різним пристроям і програмам легко інтерпретувати отримані повідомлення та використовувати їх без додаткових перетворень [3], [4].

У такій інфраструктурі велике значення має спосіб, у який додаток опрацьовує отриманий вміст. В одному випадку це може бути відображення текстового повідомлення з позначенням автора та часу надсилання, в іншому — негайне реагування пристрою на контрольний сигнал. Відповідно до намірів розробників або потреби користувачів, корисне навантаження може містити різноманітні елементи: текст, службові інструкції, потоки даних, мультимедіа. Після надходження цієї інформації програма здатна реконструювати потрібний контекст, зобразити додаткові атрибути або запустити певну дію [3], [4].

Реалізація такої технології дає змогу формувати середовище, де швидкий обмін інформацією перетворюється на основу всієї взаємодії, починаючи від простого текстового спілкування та завершуючи складними сценаріями управління пристроями або системами. Завдяки структурованості даних, можливості підтримувати стійкі канали зв’язку та гнучкому

інструментарію інтеграції, API для обміну повідомленнями в реальному часі надають необхідні механізми для створення сучасних систем, які працюють без збоїв у динамічних середовищах та здатні масштабуватися відповідно до навантаження [3], [4].

Порівняння механізмів взаємодії в реальному часі з вебхуками можна описати через відмінності в принципах їх роботи. Вебхуки реагують на події, надсилаючи дані за заздалегідь визначеними адресами, що підходить для точкових інтеграцій або запуску окремих дій на стороні сервера. Проте такий підхід не створений для складних схем спілкування між великою кількістю учасників, оскільки усі повідомлення спрямовуються на фіксовані кінцеві точки, а взаємодія не передбачає широкого розповсюдження даних. Натомість у системах обміну інформацією в реальному часі використовується модель, де повідомлення можуть розсилатися багатьом отримувачам одночасно, без потреби визначати їх наперед. Через це подібні рішення краще підходять для багатокористувацьких застосунків, у яких події мають миттєво досягати всіх зацікавлених сторін [4], [5].

Реалізувати механізм передавання інформації у режимі реального часу можна різними шляхами. Один із варіантів полягає у створенні власної інфраструктури на основі протоколів, що підтримують постійне з’єднання та швидке передавання даних. До таких технологій належать XMPP, WebSockets або механізми періодичного довготривалого опитування, здатні забезпечити гнучку та масштабовану взаємодію. Проте використання цих рішень породжує великий обсяг додаткових завдань, серед яких утримання серверної частини, підтримка стабільного з’єднання користувачів, контроль навантаження, розподіл даних, реплікація, забезпечення доступності та багато інших аспектів, що вимагають окремих ресурсів і значного технічного досвіду. Такий підхід на практиці потребує великого вкладення часу й часто відволікає від розроблення ключових функцій самого продукту [4], [5].

З огляду на це значно ефективніше спиратися на готові інструменти, що надають можливість інтегрувати обмін повідомленнями у реальному часі

через API. Сервіси цього типу знімають із розробників більшість мережевих проблем, дозволяючи зосередитись насамперед на функціональних можливостях застосунку. Завдяки готовим інструментам стає простішим не лише додавання нових компонентів, а й підтримання стабільної роботи, оскільки масштабування, швидкість передавання даних, керування з'єднаннями та оброблення запитів відбуваються на стороні сервісу. Таким чином, взаємодія зі складними механізмами передавання інформації перетворюється на роботу з доступним для інтеграції API [4], [5].

Під час вибору платформи, що надає можливість вбудувати зв'язок у реальному часі, постає питання відповідності сервісу технічним потребам. Важливо врахувати, наскільки просто інтегрується API, чи дозволяє він швидко створювати нові функції, чи працює з уже наявними технологіями, чи здатен витримувати навантаження та забезпечувати стабільну швидкодію у будь-яких умовах. Від рішень із гнучкими API залежить те, чи зможе команда легко будувати складні сценарії взаємодії та наскільки ефективно можна буде під'єднувати сторонні інструменти або додаткові сервіси [4], [5].

Підтримка SDK також має велике значення, адже саме вони забезпечують узгодженість між платформами, мовами програмування та різними середовищами розгортання. Надійний набір інструментів дозволяє створювати застосунки для веба, мобільних платформ, IoT-пристроїв та ігор, не турбуючись про внутрішню інфраструктуру. Використання таких SDK допомагає швидко налаштувати передачу повідомлень у реальному часі, оптимізувати взаємодію між частинами системи та зосередитися на розбудові функцій, які підвищують якість користувацького досвіду [5], [6].

Функціонування чатів у режимі безперервної взаємодії ґрунтується на підтриманні активного каналу зв'язку між пристроєм користувача та серверною частиною системи. Для цього застосовуються протоколи, здатні забезпечувати сталий обмін даними без необхідності повторних запитів. Через такий канал інформація миттєво передається від відправника до отримувачів, після чого відразу відображається в інтерфейсі, що створює ефект постійної

присутності та живої комунікації. Подібна схема роботи базується на взаємодії декількох компонентів, які відповідають за відтворення інтерфейсу, підтримання зв’язку, маршрутизацію повідомлень та зберігання історії діалогів. Завдяки цьому користувач отримує можливість надсилати та бачити повідомлення без відчутних затримок, а сама система забезпечує синхронність дій усіх учасників [5], [6].

Щоб підтримувати безперебійну роботу, платформи такого типу об’єднують клієнтську частину, що відповідає за візуальну взаємодію, сервер, який регулює доставку та управління повідомленнями, постійний канал передачі даних та сховище, де зберігається історія обміну. Крім базового передавання тексту, такі системи обробляють події, що виникають під час взаємодії, зокрема реакції користувача, надходження нових даних, зміну статусу підключення, відображення введення та доставку вмісту для тих, хто перебував поза мережею. Завдяки цьому комунікація відбувається плавно навіть у складних сценаріях [5], [6].

У більшості рішень цього типу реалізовано можливість обміну повідомленнями всередині програмного застосунку, що дозволяє користувачам підтримувати швидку взаємодію незалежно від кількості учасників. Сучасні програмні платформи часто охоплюють різноманітні формати даних, включаючи мультимедійний контент. Окрім тексту, надсилаються файли, зображення, аудіо, відео, а також здійснюється екранний обмін або передавання потоків у реальному часі. Паралельно з цим користувач постійно отримує індикатори подій, що сигналізують про нові повідомлення або активність інших учасників. Інтерфейс у таких програмних системах зазвичай гнучкий та реагує на потреби користувача, дозволяючи змінювати зовнішній вигляд, структуру діалогів або порядок відображення інформації. Важливо також те, що сучасні платформи намагаються забезпечити доступність для людей з особливими потребами, застосовуючи функціонал, що спрощує навігацію та роботу з інтерфейсом [5], [6].

Сфера застосування технологій синхронного спілкування охоплює широкий набір сценаріїв. Такі рішення використовуються в інструментах взаємодії між співробітниками, у соціальних платформах, у системах для організації спільної гри, у службах підтримки, у сервісах трансляцій та в застосунках, створених для встановлення особистих контактів. Розвиток штучного інтелекту сприяв появі автоматизованих співрозмовників, здатних відповідати на запити в той момент, коли користувачеві це потрібно. Такі системи дозволяють оперативно реагувати на запити, обробляти стандартні звернення та забезпечувати доступ до інформації у будь-який час. Подібний підхід значно підвищує якість взаємодії та допомагає скоротити час очікування відповіді [5], [6].

Програмні месенджери стали універсальним середовищем для обміну інформацією та координації дій. Вони забезпечують стабільне спілкування як у приватних діалогах, так і в великих групах, що зручно для командної роботи чи повсякденного спілкування. Той самий принцип застосовується в сервісах трансляцій: інтегрований чат створює простір для обговорень, реакцій та взаємодії між аудиторією та автором потоку. У системах відеоконференцій чат доповнює голосове спілкування, дозволяючи обмінюватися думками без переривання основної розмови. Усі ці приклади демонструють, що обмін повідомленнями у реальному часі став ключовим механізмом сучасної цифрової взаємодії, забезпечуючи швидкість, залучення та природність комунікації [6], [7].

Під час додавання чату до програмного забезпечення зазвичай постає питання про те, чи варто створювати таку систему самостійно, чи доцільніше інтегрувати готові рішення. Реалізація власного модуля надає повну свободу у виборі функціональності та архітектури, однак потребує значних зусиль для налаштування механізмів синхронізації, доставки, захисту та подальшого масштабування. Використання зовнішніх API суттєво скорочує час розробки, оскільки постачальники пропонують стабільні та перевірені інструменти з уже

передбаченими можливостями модерації, відповідності вимогам і продуктивної роботи на різних платформах [6], [7].

Основою взаємодії в чатах реального часу є постійні канали обміну, які реалізуються переважно на WebSocket-з'єднаннях. Цей механізм забезпечує безперервний зв'язок між сервером і клієнтом. Хоча розробка власного WebSocket-рішення можлива на будь-якій серверній мові, на практиці частіше використовуються бібліотеки на кшталт Socket.IO, що спрощують виконання складних технічних задач [6], [7].

Організація бекенд-частини відіграє ключову роль, оскільки саме вона забезпечує синхронність даних, облік історії повідомлень, керування користувачами та доступами. Для зберігання інформації зазвичай застосовуються популярні бази даних, зокрема PostgreSQL, MariaDB, MongoDB або сервіси на зразок Firebase Realtime Database, що добре підходять для невеликих систем. Масштабування часто ґрунтується на використанні кешів і брокерів повідомлень, які керують потоками даних і забезпечують безперебійність роботи навіть за зростання навантаження [6], [7].

Розгортання подібних сервісів зазвичай здійснюється на хмарних платформах, що дозволяють автоматично адаптувати обчислювальні ресурси залежно від інтенсивності використання. Такі середовища дають можливість підтримувати стабільність роботи та передбачають широкий набір інструментів для моніторингу [6], [7].

Окрему увагу під час створення чатів приділяють взаємодії з користувачем. Інтерфейс має залишатися простим, логічно впорядкованим та однаково комфортним як на мобільних пристроях, так і в браузерних версіях. Узгоджені кольори, шрифти та адаптивність допомагають уникати плутанини й підтримують відчуття цілісності [6], [7].

Програмні застосунки для обміну повідомленнями зазвичай включають численні інтерактивні деталі, що створюють відчуття безперервного спілкування: від індикаторів набору до анімації та реакцій. Частина таких

елементів має усталений вигляд, у той час як інші залишають простір для творчості дизайнерів, які орієнтуються на вподобання аудиторії [6], [7].

Безперервність обміну та швидкість реакції базуються на поєднанні оптимізованих протоколів, кешування та мереж доставки контенту. Завдяки таким технологіям можливе швидке завантаження та мінімізація затримок навіть за високого трафіку, а також забезпечується доступ до повідомлень у випадках, коли мережеве з'єднання тимчасово недоступне [6], [7].

Питання захисту даних стає визначальним, оскільки через чат проходить значна частина особистої інформації. Для гарантування конфіденційності впроваджується шифрування, організовується захищене зберігання та періодично проводяться перевірки безпеки. Користувачу має надаватися можливість самостійно управляти своїми даними: приховувати контакти, обмежувати взаємодію, чистити історію або повністю видаляти обліковий запис [6], [7].

Визначено, що програмні засоби для передачі повідомлень у реальному часі забезпечують миттєву взаємодію між користувачами та дозволяють інформації надходити без затримок. Таке програмне забезпечення здатне обробляти повідомлення відразу після їхнього створення, що робить спілкування максимально природним і наближеним до живого діалогу. Подібні програмні рішення створюються для того, щоб підтримувати постійний потік даних між учасниками комунікації, незалежно від їхнього місцезнаходження чи типу пристрою..

1.2 Аналіз програмного забезпечення для передачі повідомлень у реальному часі

Проаналізуємо програмне забезпечення для передачі повідомлень у реальному часі [8]-[15].

Програмне забезпечення Connecteam (рис. 1.1) є універсальною платформою, що поєднує миттєвий обмін повідомленнями з інструментами

для організації робочих процесів і взаємодії між співробітниками, які виконують свої обов'язки поза офісним середовищем. У системі створено середовище, де комунікація, управління робочими змінами та контроль виконання завдань утворюють єдину узгоджену структуру, здатну підтримувати ефективність команди незалежно від місця виконання роботи. Такий підхід дає змогу підтримувати постійний зв'язок із персоналом і оперативно реагувати на зміни, підвищуючи рівень інформованості та продуктивності [8], [13].

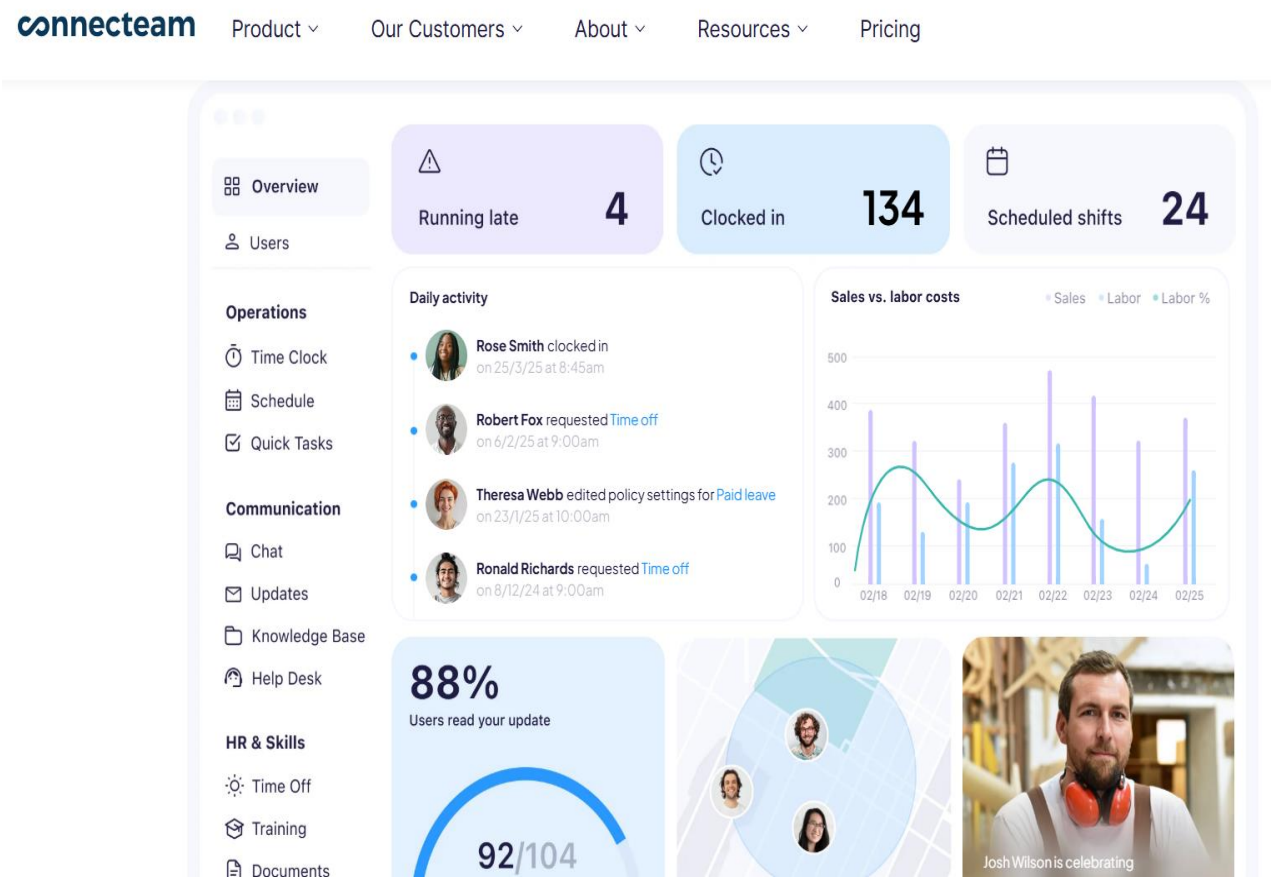

Рисунок 1.1 – Програмне забезпечення Connecteam [13]

Функціональність програми Connecteam забезпечує передачу повідомлень без затримок, що дозволяє уникнути надмірної залежності від електронної пошти та скорочує час на координацію дій. У межах одного застосунку зосереджено інструменти для створення розкладів, ведення обліку робочих змін, фіксації робочого часу та контролю виконання доручень.

Платформа підтримує можливість надання навчальних матеріалів та проведення опитувань, що підсилює залученість співробітників і сприяє кращому засвоєнню інформації, необхідної для виконання службових обов'язків [8], [13].

Програмне забезпечення Connecteam підтримує створення гнучких робочих процесів, що можуть адаптуватися до потреб конкретної організації, а її мобільно орієнтований інтерфейс дає змогу використовувати весь функціонал під час пересування. Застосування Connecteam є особливо корисним у сферах, де значна частина персоналу працює поза традиційним офісом і потребує зручного інструмента для об'єднання комунікаційних та операційних завдань [8], [13].

Платформа працює на різних пристроях, включно з мобільними та вебсередовищем, що забезпечує безперервний доступ до інструментів незалежно від місця перебування. Можливість обирати тарифний план відповідно до кількості користувачів та потрібних функцій робить систему придатною як для невеликих колективів, так і для великих організацій із розгалуженою структурою [8], [13].

Основними характеристиками програмного забезпечення Connecteam є такі [8], [13]:

– управління змінами розкладу: програмне забезпечення Connecteam дозволяє швидко створювати, редагувати та розсилати графіки роботи співробітникам у режимі реального часу [8], [13];

– інструменти комунікації: програмне забезпечення Connecteam забезпечує корпоративний чат, оновлення, оголошення та пости для оперативної взаємодії між працівниками [8], [13];

– відстеження робочого часу: програмне забезпечення Connecteam дає можливість фіксувати години роботи за допомогою мобільного таймера, GPS-міток і автоматичних нагадувань [8], [13];

– керування завданнями: програмне забезпечення Connecteam дозволяє створювати та призначати задачі, контролювати їх виконання та отримувати звіти щодо прогресу [8], [13];

– навчальні модулі: програмне забезпечення Connecteam надає інструменти для створення інструкцій, політик, курсів і перевірок знань співробітників [8], [13];

– форми та контрольні списки: програмне забезпечення Connecteam підтримує створення електронних форм, чек списків і звітів для автоматизації щоденних операцій [8], [13];

– інтеграції з іншими сервісами: програмне забезпечення Connecteam підтримує підключення до платіжних систем, бухгалтерських платформ та HR-рішень [8], [13];

– аналітика та звітність: програмне забезпечення Connecteam забезпечує збір даних, формування статистики та аналітичних звітів для керування персоналом [8], [13].

Серед недоліків програмного забезпечення Connecteam можна відзначити обмежені можливості безкоштовного тарифу, складність налаштування деяких модулів для нових користувачів, а також те, що частина функцій орієнтована переважно на мобільне використання, що може бути незручним для компаній, які потребують розширеного настільного інтерфейсу [8], [13].

Програмне забезпечення Quidget (рис. 1.2) використовується як інструмент для миттєвої комунікації з відвідувачами сайтів та користувачами сервісів і працює на основі інтелектуальних алгоритмів, здатних автоматично обробляти поширені звернення. Система функціонує як інтерактивний помічник, який бере на себе виконання стандартних інформаційних запитів і тим самим знижує навантаження на службу підтримки. Завдяки такому підходу скорочується час очікування відповіді, а робота команди стає результативнішою [9], [14].

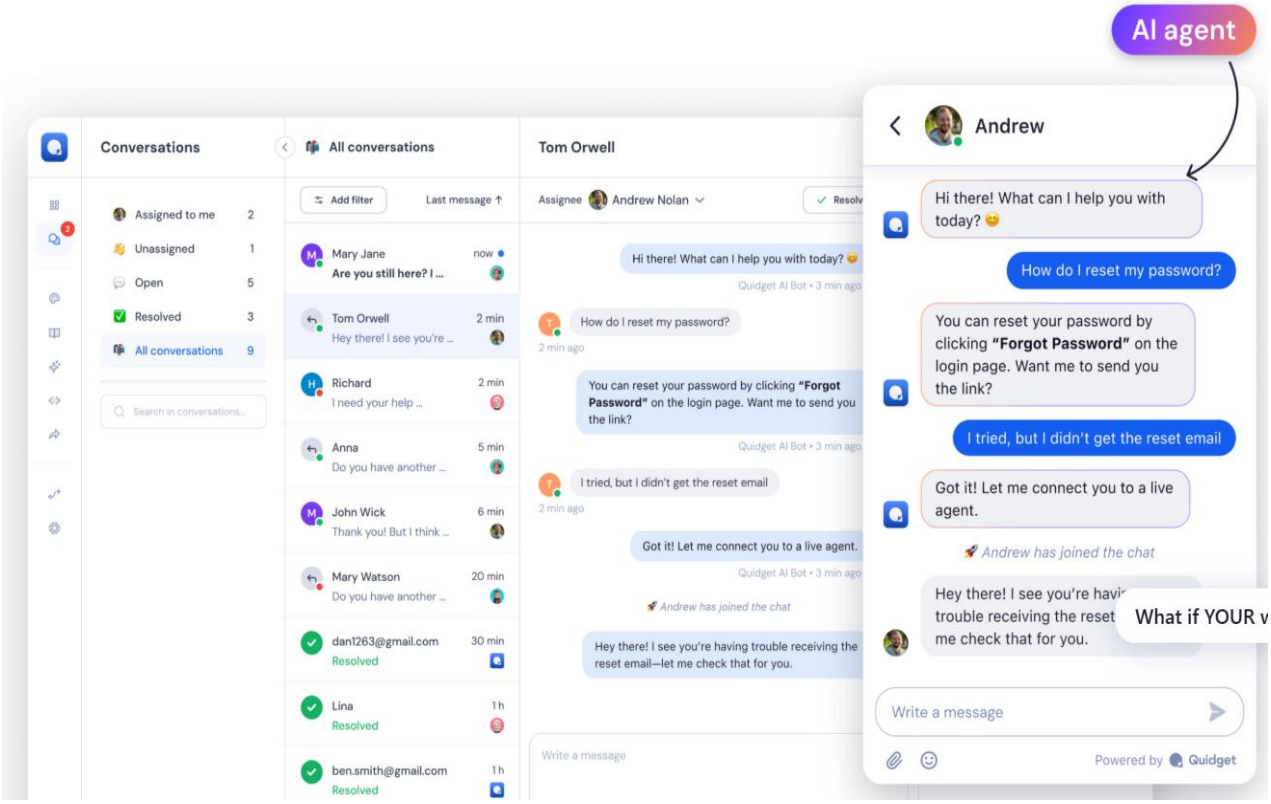

Рисунок 1.2 – Програмне забезпечення Quidget [14]

Однією з ключових можливостей платформи є передавання складних звернень консультантам, коли автоматичний обробник не може надати достатньо точну відповідь. Це дозволяє поєднувати швидкість машинної обробки зі здатністю людини розв’язувати індивідуальні ситуації, що підвищує якість взаємодії зі споживачами. Оскільки рутинні питання вирішуються автоматично, персонал може приділяти більше уваги завданням, які потребують глибшого аналізу [9], [14].

Важливою перевагою Quidget є підтримка великої кількості мов, що дає змогу використовувати його компаніям, орієнтованим на міжнародну аудиторію. Користувачі мають можливість спілкуватися тією мовою, яка їм зручніша, що позитивно впливає на рівень сервісу та робить інструмент придатним для різних ринків. Просте встановлення без технічних вимог дозволяє швидко інтегрувати платформу з популярними системами керування контентом та Інтернет магазинами, тому розгортання не потребує спеціальної підготовки [9], [14].

Програмне забезпечення Quidget також поєднується з різними сторонніми сервісами для роботи із заявками, внутрішньою комунікацією та месенджерами, що дає змогу налаштовувати єдиний простір для взаємодії з користувачами на різних каналах. Система здатна навчатися на наявній інформації компанії, що підвищує точність відповідей і сприяє більш ефективному обслуговуванню [9], [14].

Разом з тим програмне забезпечення Quidget має певні обмеження. Його основна спрямованість на автоматизацію не завжди забезпечує гнучке втручання оператора, а можливості індивідуального налаштування алгоритмів опрацювання природної мови можуть виявитися недостатніми для складних бізнес-сценаріїв [9], [14].

Програмне забезпечення Quidget має такі особливості [9], [14]:

– автоматизована обробка звернень: програмне забезпечення Quidget самостійно відповідає на типові запитання користувачів, зменшуючи навантаження на службу підтримки та скорочуючи час реагування [9], [14];

– передавання складних випадків оператору: у разі нестандартних ситуацій чат передає розмову людині, що дає змогу забезпечити більш точні та персоналізовані відповіді [9], [14];

– підтримка багатьох мов: програмне забезпечення Quidget працює з широким спектром мов, що робить його придатним для компаній, орієнтованих на міжнародну аудиторію [9], [14];

– відсутність потреби у програмуванні: розгортання та налаштування програмного забезпечення Quidget виконуються без технічних знань, що дає можливість швидко інтегрувати інструмент у робоче середовище [9], [14];

– сумісність із популярними платформами: програмне забезпечення Quidget легко підключається до сервісів для підтримки клієнтів, корпоративних месенджерів та CMS-платформ, формуючи єдину інфраструктуру взаємодії [9], [14];

– навчання на базі знань компанії: чат-бот Quidget адаптує відповіді, опираючись на інформацію, надану організацією, що підвищує точність та корисність комунікації [9], [14].

Серед недоліків програмного забезпечення Quidget можна відзначити його залежність від стабільного Інтернет з'єднання, обмеження деяких функцій у базових тарифах та можливу складність під час адаптації для великих організацій із розгалуженою структурою процесів [9], [14].

Програмне забезпечення MirrorFly (рис. 1.3) є універсальним рішенням для створення комунікаційних можливостей у цифрових продуктах. Програмне забезпечення MirrorFly дає змогу інтегрувати текстове спілкування, голосові взаємодії та відеозв'язок без необхідності створювати ці механізми з нуля [10], [15].

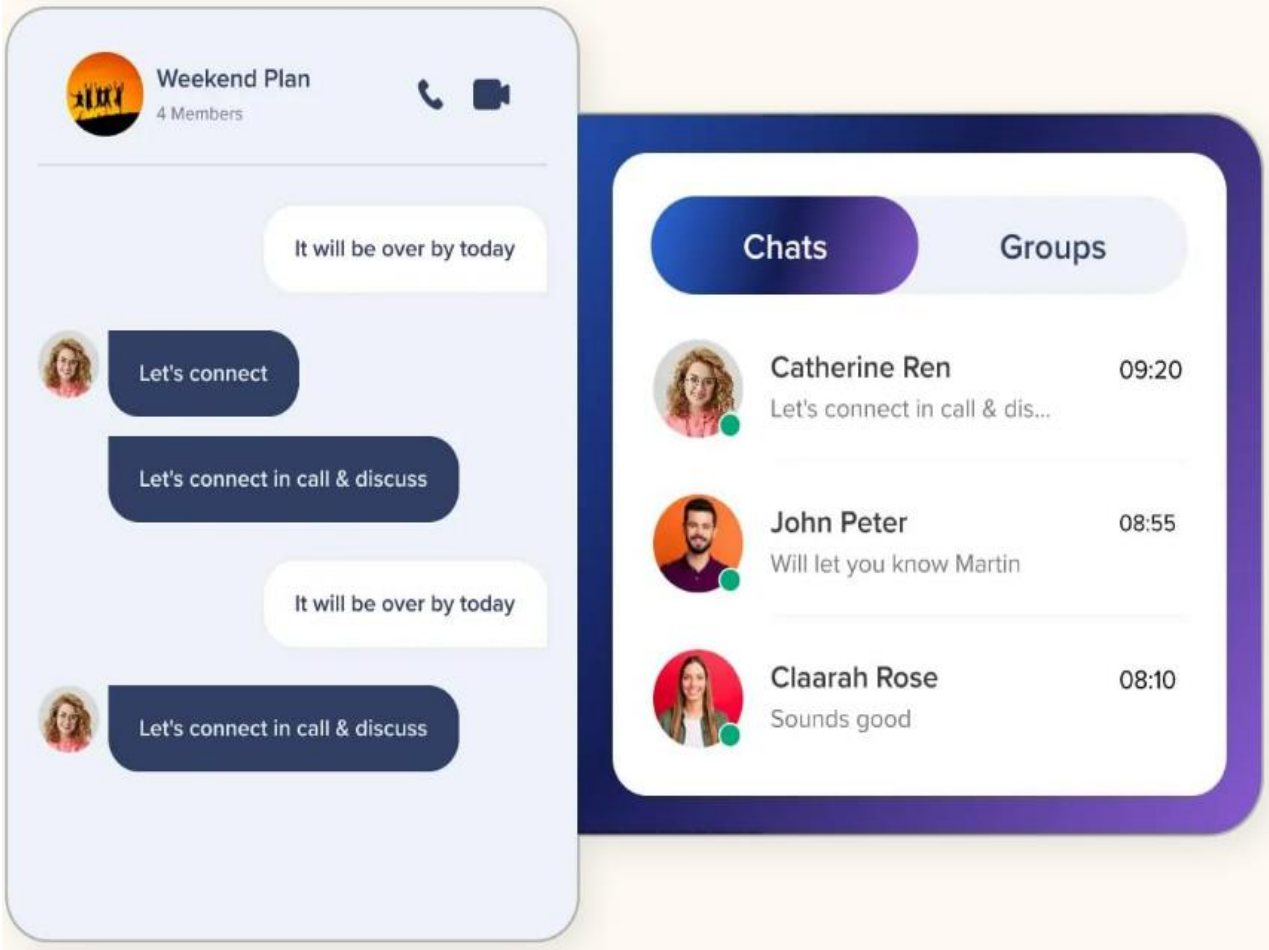

Рисунок 1.3 – Програмне забезпечення MirrorFly [15]

У межах цього сервісу пропонуються інструменти, що дозволяють розробникам вбудовувати модулі реального часу у мобільні додатки або вебплатформи, забезпечуючи стабільну роботу та високу якість передавання інформації [10], [15].

Система MirrorFly орієнтована на масштабованість, тому підтримує значну кількість одночасних розмов, незалежно від того, чи йдеться про індивідуальні діалоги, групове листування чи дзвінки у великі команди. Технологія забезпечує можливість миттєвої зміни мови під час текстової взаємодії, що сприяє плавному спілкуванню між користувачами з різних країн. Окрім текстового обміну, платформа підтримує передачу файлів, показ екрана та надсилання даних про місцезнаходження, що підсилює функціональність та робить систему придатною для різних сценаріїв використання — від бізнес-комунікацій до внутрішніх корпоративних сервісів [10], [15].

Платформа MirrorFly працює на широкому спектрі пристроїв і операційних систем, охоплюючи настільні та мобільні середовища. Завдяки цьому розробники можуть створювати продукти, які функціонують узгоджено на різних типах техніки, забезпечуючи користувачам однаковий рівень зручності та стабільності незалежно від того, з якого пристрою вони входять у систему [10], [15].

Програмне забезпечення MirrorFly має такі особливості [10], [15]:

– підтримка різних каналів зв’язку: програмне забезпечення MirrorFly забезпечує об’єднання текстових повідомлень, голосових дзвінків та відеоспілкування в одному рішенні [10], [15];

– масштабованість платформи: програмне забезпечення MirrorFly здатне витримувати значні навантаження та підтримувати велику кількість одночасних чатів і дзвінків без втрати якості [10], [15];

– переклад у режимі реального часу: надається можливість автоматичного перетворення текстових повідомлень на різні мови під час активної взаємодії користувачів [10], [15];

– розширені можливості обміну: у програмному забезпеченні MirrorFly підтримується передавання файлів, демонстрація екрана та надсилання інформації про поточне розташування [10], [15];

– сумісність з різними платформами: програмне забезпечення MirrorFly стабільно працює на широкому переліку операційних систем і пристроїв, забезпечуючи рівномірний користувацький досвід [10], [15].

Серед недоліків програмного забезпечення MirrorFly можна відзначити те, що розширені можливості персоналізації можуть вимагати додаткової технічної підготовки, інтеграція передових функцій інколи потребує ретельного налаштування інфраструктури [10], [15].

Порівняльну характеристику програмного забезпечення для передачі повідомлень у реальному часі наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння програмного забезпечення для передачі повідомлень у реальному часі

Критерій порівняння	Connecteam [13]	Quidget [14]	MirrorFly [15]
Підтримка багатьох мов	+–	+	+
Масштабованість	+	+–	+
Орієнтація на внутрішню комунікацію	+	–	+–
Орієнтація на підтримку клієнтів	+–	+	+–
Гнучкість налаштування	+–	+–	+
Підходить для включення в інші застосунки	–	+–	+

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить багато програмних засобів для передачі повідомлень у реальному часі. Проте деякі програмні засоби для передачі повідомлень у

реальному часі можуть не забезпечувати стабільну роботу в умовах високого навантаження. У таких програмних системах навіть незначні збої можуть призвести до втрати важливих повідомлень або затримки їх доставки, що одразу впливає на користувацький досвід і порушує робочі процеси. Залежність від швидкості мережі створює додаткові ризики, оскільки будь-які коливання пропускної здатності чи нестабільність з'єднання миттєво позначаються на якості передавання даних. Проблемою залишається і забезпечення безпеки. Передавання даних у реальному часі потребує постійного шифрування та захисту каналів зв'язку, а також контролю за цілісністю даних у момент їх передачі. Будь-які помилки у конфігурації або недосконаліть механізмів аутентифікації створюють потенційні точки доступу для зловмисників. Поєднання високої швидкості роботи та необхідності негайної реакції іноді зменшує можливість застосування надто громіздких механізмів захисту, що збільшує ризики витоку інформації. У багатьох випадках користувачі змушені довіряти стороннім провайдерам критично важливу інформацію, що не завжди відповідає їхнім вимогам до приватності. Тому актуальною є розробка програмного забезпечення для передачі повідомлень у реальному часі.

При розробці програмного забезпечення для передачі повідомлень у реальному часі необхідно забезпечити такі функціональні вимоги:

– підтримка можливості авторизації до системи передачі повідомлень;

– підтримка можливості відображення списку користувачів, доступних до спілкування;

– можливість виведення списку текстових повідомлень, що відображають процес спілкування;

– можливість відправлення та отримання повідомлень для спілкування з іншими користувачами;

– підтримка можливості використання спеціальних графічних символів (смайли) у процесі спілкування;

– підтримка можливості зміни кольорової гами в інтерфейсі користувача.

1.3 Висновки за розділом 1

Визначено, що програмні засоби для передачі повідомлень у реальному часі забезпечують миттєву взаємодію між користувачами та дозволяють інформації надходити без затримок. Таке програмне забезпечення здатне обробляти повідомлення відразу після їхнього створення, що робить спілкування максимально природним і наближеним до живого діалогу. Подібні програмні рішення створюються для того, щоб підтримувати постійний потік даних між учасниками комунікації, незалежно від їхнього місцезнаходження чи типу пристрою.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для передачі повідомлень у реальному часі. Проте деякі програмні засоби для передачі повідомлень у реальному часі можуть не забезпечувати стабільну роботу в умовах високого навантаження. Проблемою залишається і забезпечення безпеки. Передавання даних у реальному часі потребує постійного шифрування та захисту каналів зв’язку, а також контролю за цілісністю даних у момент їх передачі. У багатьох випадках користувачі змушені довіряти стороннім провайдерам критично важливу інформацію, що не завжди відповідає їхнім вимогам до приватності. Тому актуальною є розробка програмного забезпечення для передачі повідомлень у реальному часі.

Сформульовано функціональні вимоги до програмного забезпечення для передачі повідомлень у реальному часі.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

При розробці програмного забезпечення для передачі повідомлень у реальному часі було використано мову програмування Java [16], [17].

Мова програмування Java вирізняється особливою увагою до архітектурної дисципліни. Її структура спонукає до впорядкованого проєктування, а сама парадигма орієнтована на передбачуваність поведінки. Такий підхід формує середовище, де логіка системи стає прозорішою, а помилки легше відстежуються. У великих проєктах, де працюють десятки або сотні розробників, це створює додаткову цінність, оскільки уніфікованість стилю сприяє покращенню колективної роботи. Мова дозволяє уникати хаосу, характерного для деяких технологій з менш суворою структурною організацією [16], [17].

Окремої уваги заслуговує розвиток екосистеми мови програмування Java, що постійно оновлюється, зберігаючи сумісність і стабільність. Це підтримує довіру з боку бізнесу, який прагне уникати ризиків, пов’язаних із радикальними змінами. Навіть коли з’являються нові інструменти й підходи, вони інтегруються обережно, щоб зберегти можливість безболісного переходу з попередніх версій. Така стратегія робить мову програмування Java значущою у сферах, де передбачуваність має не менше значення, ніж технічна досконалість [16], [17].

Мова програмування Java демонструє здатність поєднувати консервативність із відкритістю до розвитку. Мова програмування Java зберігає принципи, що були закладені на початку, але водночас отримує механізми, які дозволяють відповідати вимогам сучасних розподілених систем, хмарних платформ і обчислювальних кластерів. Її підхід полягає не у швидкому переосмисленні основ, а у поступовому посиленні та розширенні функціональності, що дозволяє досягати балансу між стабільністю й інноваційністю [16], [17].

Значення Java також підсилюється її здатністю працювати в середовищах, де цінується контрольованість поведінки та детермінованість процесів. У фінансовому секторі, телекомунікаціях, державних структурах і великих корпоративних системах саме така властивість забезпечує довготривалу довіру до мови. У цих умовах Java сприймається не як модний інструмент, а як технологічний фундамент, на якому можна будувати системи, розраховані на десятиліття [16], [17].

Вибір мови Java для створення програмного забезпечення, що забезпечує передавання повідомлень у реальному часі, пояснюється поєднанням стабільності, продуктивності та передбачуваності, які ця технологія забезпечує на рівні архітектури та інструментарію. У середовищі, де затримки мають бути мінімальними, а системи повинні працювати безперервно, значення має кожен механізм керування ресурсами, і Java надає розробнику інструменти, що дають змогу підтримувати таку роботу без надмірних зусиль [16], [17].

У застосунках реального часу критично важливо мати кероване середовище виконання, здатне стабільно поводитися при великих навантаженнях. Віртуальна машина Java пропонує саме таку модель, де управління пам’яттю відбувається автоматично, а поведінка програми залишається передбачуваною навіть за інтенсивного трафіку. Механізми оптимізації віртуальної машини зменшують кількість помилок, пов’язаних із ручним керуванням ресурсами, і забезпечують можливість тривалого безперервного функціонування. Завдяки цьому комунікаційні модулі, які повинні працювати постійно, мають більшу стійкість і меншу залежність від людського фактора [16], [17].

Значну роль відіграє і можливість масштабування. У системах обміну повідомленнями збільшення кількості користувачів або каналів зв’язку відбувається нерівномірно, тому програмне забезпечення повинно вміти адаптуватися до навантаження без перезавантаження чи ручного втручання. Екосистема мови програмування Java пропонує широкий набір рішень для

побудови високонавантажених серверних платформ, де підтримка потоків, асинхронних операцій і неблокувальних механізмів дозволяє ефективно розподіляти обчислювальну роботу. Це створює передумови для побудови програмних сервісів, які залишаються чутливими навіть при значному навантаженні [16], [17].

У сфері реального часу важливо мати доступ до перевірених протоколів та інструментів, що забезпечують швидке з'єднання, синхронізацію та керування подіями. Java має розвинену базу бібліотек і фреймворків, що підтримують роботу з вебсокетами, потоковими даними, брокерами повідомлень і хмарними сервісами. Завдяки цьому розробник отримує можливість швидко інтегрувати існуючі рішення та зосередитися на логіці продукту, а не на реалізації низькорівневих протоколів [16], [17].

Суттєве значення має і кросплатформність. Програмне забезпечення, яке передає повідомлення в реальному часі, часто повинно працювати в різних середовищах, від локальних серверів до хмарних кластерів. Java дозволяє запускати ті самі модулі на будь-якій платформі без модифікації коду, що спрощує розгортання і зменшує витрати на підтримку [16], [17].

Важливою перевагою мови програмування Java є високий рівень безпеки. Обмін повідомленнями в режимі реального часу пов'язаний зі значним ризиком перехоплення чи втручання, а Java пропонує широкий спектр вбудованих механізмів шифрування, керування доступом і контролю цілісності. Це дає змогу створювати системи, які поєднують швидку реакцію з надійним захистом [16], [17].

Завдяки сукупності цих характеристик Java забезпечує надійну основу для програмних систем, що працюють у режимі реального часу. Вона поєднує стабільність, масштабованість, безпеку та багаторічний досвід використання в критично важливих рішеннях, що робить її доцільним вибором для розробки програмного забезпечення, орієнтованого на швидкий, безперервний і керований обмін повідомленнями [16], [17].

Обґрунтування вибору мови програмування для розробки програмного забезпечення для передачі повідомлень у реальному часі наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для розробки програмного забезпечення для передачі повідомлень у реальному часі

Критерій порівняння мов програмування	Мова програмування		
	Java	JavaScript	Python
Продуктивність під високим навантаженням	+	+–	–
Масштабованість серверних застосунків	+	+–	+–
Розвиненість інструментів для багатопотоковості	+	–	+–
Наявність промислових рішень для обробки подій у реальному часі	+	+–	+–
Рівень безпеки та вбудованих механізмів захисту	+	+–	+–

Отже, для реалізації програмного забезпечення для передачі повідомлень у реальному часі обрано мову програмування Java, яка поєднує стабільність, безпеку та багаторічний досвід використання в критично важливих програмних рішеннях. Крім того, екосистема Java пропонує широкий набір програмних рішень для побудови високонавантажених серверних платформ, де підтримка потоків, асинхронних операцій і неблокувальних механізмів дозволяє ефективно розподіляти обчислювальну роботу. Це створює передумови для побудови програмних сервісів, які залишаються чутливими навіть при значному навантаженні.

2.2 Вибір середовища розробки для створення програмного забезпечення для передачі повідомлень у реальному часі

В якості середовища розробки для створення програмного забезпечення для передачі повідомлень у реальному часі було обрано середовище Eclipse [18], [19].

Середовище розробки Eclipse є надає гнучкий простір для створення програмного забезпечення. Особливістю Eclipse є здатність забезпечувати майже лабораторні умови для відлагодження. Розробник отримує можливість зосередитися не лише на написанні коду, але й на аналізі його поведінки в динаміці, спостерігаючи за внутрішніми процесами так, ніби система розгорнута у виробничому середовищі. Такий підхід створює передумови для глибшого розуміння взаємодії модулів, що особливо набуває значення у складних застосунках, де один некоректний елемент може створити затримку чи порушити узгодженість усього механізму [18], [19].

Екосистема середовища розробки Eclipse також вирізняється тим, що залишає простір для тривалого зростання проєктів. Середовище сприймає масштабування не як винятковий сценарій, а як природну частину розвитку системи. Воно дає змогу підтримувати роботу над проєктом роками, поступово нарощувати функціональність та зберігати контроль над історією змін. Такий підхід відкриває можливість працювати з кодом так само ретельно, як із довготривалим інженерним об’єктом, у якому кожна зміна має значення [18], [19].

Важливою рисою є здатність Eclipse бути водночас універсальним і спеціалізованим інструментом. Середовище може виконувати роль платформи для звичайного текстового редагування, але в той самий час здатне перетворюватися на багатофункціональний комплекс із повною інтеграцією засобів тестування, аналізаторів, інструментів моделювання та візуалізації архітектури. Ця багатошаровість дозволяє формувати індивідуальний робочий

простір для кожного типу завдань, не розпорошуючи увагу між численними зовнішніми утилітами [18], [19].

Середовище має виразну орієнтацію на інженерну передбачуваність. Його поведінка не залежить від випадкових чинників, а логіка роботи інструментів завжди залишається чіткою та послідовною. У зв’язку з цим Eclipse часто обирається там, де необхідно уникнути експериментальних рішень і зосередитися на повторюваних, стабільних процесах, що забезпечують довгострокову підтримку й керованість [18], [19].

Не менш значущим є те, що Eclipse існує як результат співпраці великої спільноти інженерів та організацій, які підтримують розвиток його основи. Постійне оновлення не зруйнувало фундаментальних підходів, а лише зміцнило їх, надаючи середовищу здатність адаптуватися до нових стандартів і технологічних рішень. Завдяки цьому структура Eclipse залишилася цілісною, навіть у періоди, коли ринок програмного забезпечення радикально змінювався [18], [19].

У підсумку Eclipse можна охарактеризувати як інструмент, що поєднує інженерну строгість, гнучкість конфігурації, передбачуваність поведінки та культурну зрілість спільноти, яка його підтримує. Це середовище створює умови для точного, контрольованого й довговічного розвитку складних програмних систем, зокрема тих, що працюють у режимі постійної взаємодії та обміну даними [18], [19].

Використання середовища Eclipse для створення програмного забезпечення, що працює з повідомленнями в реальному часі, є доцільним завдяки поєднанню технічної зрілості, розширюваності та зручності, які це середовище надає розробнику. Eclipse формує робочий простір, орієнтований на побудову складних систем, де потрібні точність, стабільність і повний контроль над життєвим циклом застосунку [18], [19].

Середовище вирізняється тим, що дає змогу працювати з великими проєктами без відчутних обмежень у продуктивності. Структура робочого простору дозволяє чітко впорядковувати модулі, підтримувати зрозумілу

ієрархію та легко керувати численними залежностями. У системах реального часу, де важливо підтримувати дисципліну кодової бази, така організованість мінімізує ризик помилок та пришвидшує роботу багатьох розробників над одним продуктом [18], [19].

Важливу роль відіграє розвинена система плагінів. Eclipse надає широкі можливості для підключення інструментів, що оптимізують процес розробки серверних застосунків, роботу з мережевими протоколами, потоковими даними чи бібліотеками асинхронної обробки подій. Завдяки цьому середовище легко адаптується до специфіки проєктів, які вимагають швидкої реакції системи та стабільної роботи у режимі постійного обміну повідомленнями [18], [19].

Істотною перевагою є інтеграція зі стандартними для Java засобами побудови й керування залежностями. Підтримка Maven та Gradle, а також інструментів тестування, дає змогу контролювати якість коду та забезпечувати безперервну оптимізацію. Для застосунків, що працюють у режимі реального часу, це означає можливість перевіряти коректність модулів ще до їхнього запуску в експлуатацію [18], [19].

Середовище надає розширені можливості для діагностики. Інструменти профілювання дозволяють аналізувати поведінку застосунку під навантаженням, виявляти вузькі місця, відстежувати споживання ресурсів та контролювати потоки. У системах, що працюють безперервно, такі можливості є критичними, оскільки вони дозволяють утримувати мінімальну затримку та стабільну швидкість обробки повідомлень [18], [19].

Eclipse підтримує інтеграцію з серверними платформами й хмарними сервісами, що спрощує тестування систем реального часу безпосередньо з робочого середовища. Це дозволяє імітувати реальні умови функціонування, оперативно виявляти проблеми та вдосконалювати логіку обробки подій ще на етапі розробки [18], [19].

Окремим чинником є відкритість та безкоштовність, що робить середовище доступним, але водночас потужним. Відкритий код забезпечує

прозорість та гнучкість, а велика спільнота постійно створює нові розширення та вдосконалює існуючі інструменти. У проєктах реального часу, де потрібна довготривала підтримка, така стабільність екосистеми створює додаткову надійність [18], [19].

Обґрунтування вибору середовища розробки для створення програмного забезпечення для передачі повідомлень у реальному часі наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору середовища розробки для створення програмного забезпечення для передачі повідомлень у реальному часі

Критерій порівняння середовищ розробки	Середовища розробки		
	Eclipse	IntelliJ IDEA	NetBeans
Зручність роботи з багатопотоковими застосунками	+	+–	+–
Підтримка інструментів аналізу продуктивності	+	+–	+
Гнучкість конфігурації та розширюваність	+	+	+–
Інтеграція з серверними платформами	+	+	+–
Потужність інструментів відлагодження	+	+	+–

Таким чином, для створення програмного забезпечення для передачі повідомлень у реальному часі обрано середовище розробки Eclipse, що поєднує структурованість, підтримку сучасних інструментів Java, розширюваність та аналітичні можливості, необхідні для створення систем

передачі повідомлень у реальному часі. Його використання забезпечує контрольованість, ефективність та передбачуваність усіх етапів розробки.

2.3 Висновки за розділом 2

Для реалізації програмного забезпечення для передачі повідомлень у реальному часі обрано мову програмування Java, яка поєднує стабільність, безпеку та багаторічний досвід використання в критично важливих програмних рішеннях. Крім того, екосистема Java пропонує широкий набір програмних рішень для побудови високонавантажених серверних платформ, де підтримка потоків, асинхронних операцій і неблокувальних механізмів дозволяє ефективно розподіляти обчислювальну роботу. Це створює передумови для побудови програмних сервісів, які залишаються чутливими навіть при значному навантаженні.

Для створення програмного забезпечення для передачі повідомлень у реальному часі обрано середовище розробки Eclipse, що поєднує структурованість, підтримку сучасних інструментів Java, розширюваність та аналітичні можливості, необхідні для створення систем передачі повідомлень у реальному часі. Його використання забезпечує контрольованість, ефективність та передбачуваність усіх етапів розробки.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення для передачі повідомлень у реальному часі

Структуру програмного забезпечення для передачі повідомлень у реальному часі наведено на рис. 3.1.

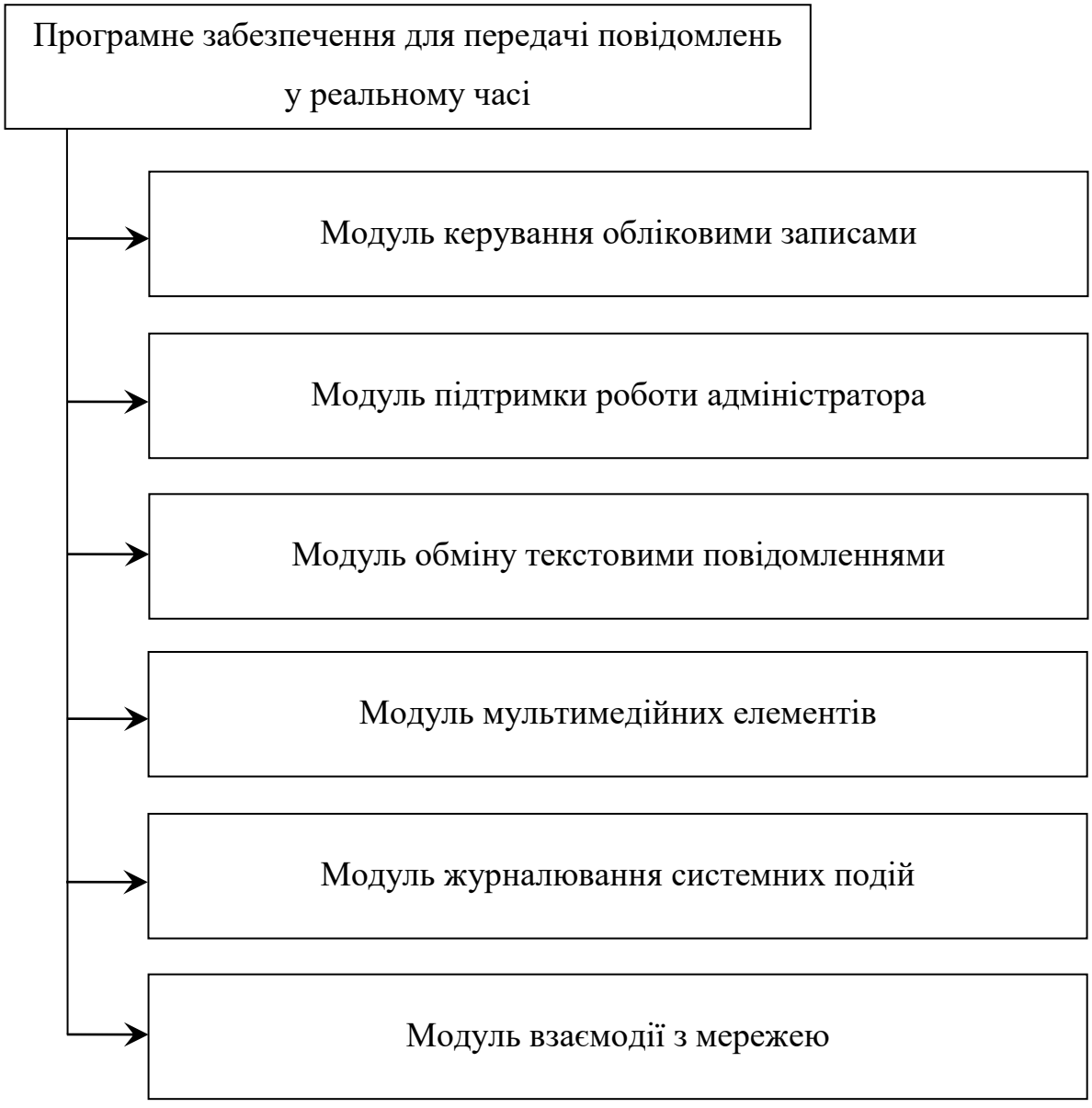

Рисунок 3.1 – Структура програмного забезпечення для передачі повідомлень у реальному часі

Архітектура програмного забезпечення, призначеного для передавання повідомлень у реальному часі, передбачає поділ на кілька взаємопов’язаних модулів, кожен з яких відповідає за окремий аспект роботи системи. Центральним елементом виступає компонент, що забезпечує керування обліковими записами. У цьому модулі зосереджено механізми створення профілів, підтвердження особи користувача та підтримання актуального стану його сеансу. Саме тут здійснюється перевірка доступу, формування дозволів та контроль над тим, хто може взаємодіяти з сервісом.

Модуль підтримки роботи адміністратора відповідає за вхід адміністратора, надає інструменти для контролю роботи системи, забезпечує доступ до журналів активності та дозволяє здійснювати втручання у разі порушення правил або технічних збоїв. Через нього підтримується порядок у системі та відстежується поведінка зареєстрованих користувачів.

Особливу роль відіграє модуль обміну текстовими повідомленнями, де реалізовано основний механізм передавання даних між учасниками комунікації. У межах цього модуля здійснюється формування повідомлення, його маршрутизація до адресата, обробка статусів доставки та забезпечення низької затримки в процесі передавання інформації. Саме він відповідає за те, щоб користувач отримував повідомлення майже миттєво.

Поряд із текстовим каналом працює модуль мультимедійних елементів. Він обробляє графічні компоненти, зокрема емодзі та зображення, забезпечує їх кодування, компресію та швидку передачу. Цей модуль гарантує, що повідомлення з візуальними елементами не створюватимуть суттєвих затримок і коректно відображатимуться на різних пристроях.

Окремим функціональним елементом виступає модуль журналювання системних подій, який накопичує дані про активність учасників, передані повідомлення та дії адміністратора. Він формує структуровану історію взаємодій, дозволяє переглядати попередні записи, а також очищувати їх, коли це необхідно для підтримання продуктивності або безпеки.

Завершує архітектуру модуль взаємодії з мережею, який забезпечує стабільний зв’язок між клієнтською частиною та сервером. Він контролює встановлення з’єднання, реагує на зміни якості мережі та впроваджує механізми повторної передачі у разі втрати пакетів. За його допомогою система зберігає працездатність у нестабільних умовах і підтримує безперервність обміну даними.

У такій структурі кожен компонент працює узгоджено, формуючи цілісну платформу, здатну забезпечити оперативне передавання повідомлень та стабільну роботу навіть при значному навантаженні.

3.2 Функціонування програмного забезпечення для передачі повідомлень у реальному часі

Функціонування програмного забезпечення для передачі повідомлень у реальному часі подамо за допомогою схеми, зображеної на рис. 3.2.

Функціонування програмного забезпечення для передачі повідомлень у реальному часі ґрунтується на послідовній взаємодії кількох механізмів, що забезпечують цілісність роботи системи та безперервність комунікації між користувачами.

Робота програмного сервісу починається з етапу ідентифікації, коли користувач вводить власні дані, а система здійснює перевірку введеної інформації та встановлює зв’язок із базою даних для підтвердження особи. Якщо обліковий запис уже існує, відбувається авторизація, якщо ні — створюється новий профіль. Такий підхід дозволяє підтримувати контроль доступу й забезпечувати персоналізовану взаємодію кожного користувача з програмою.

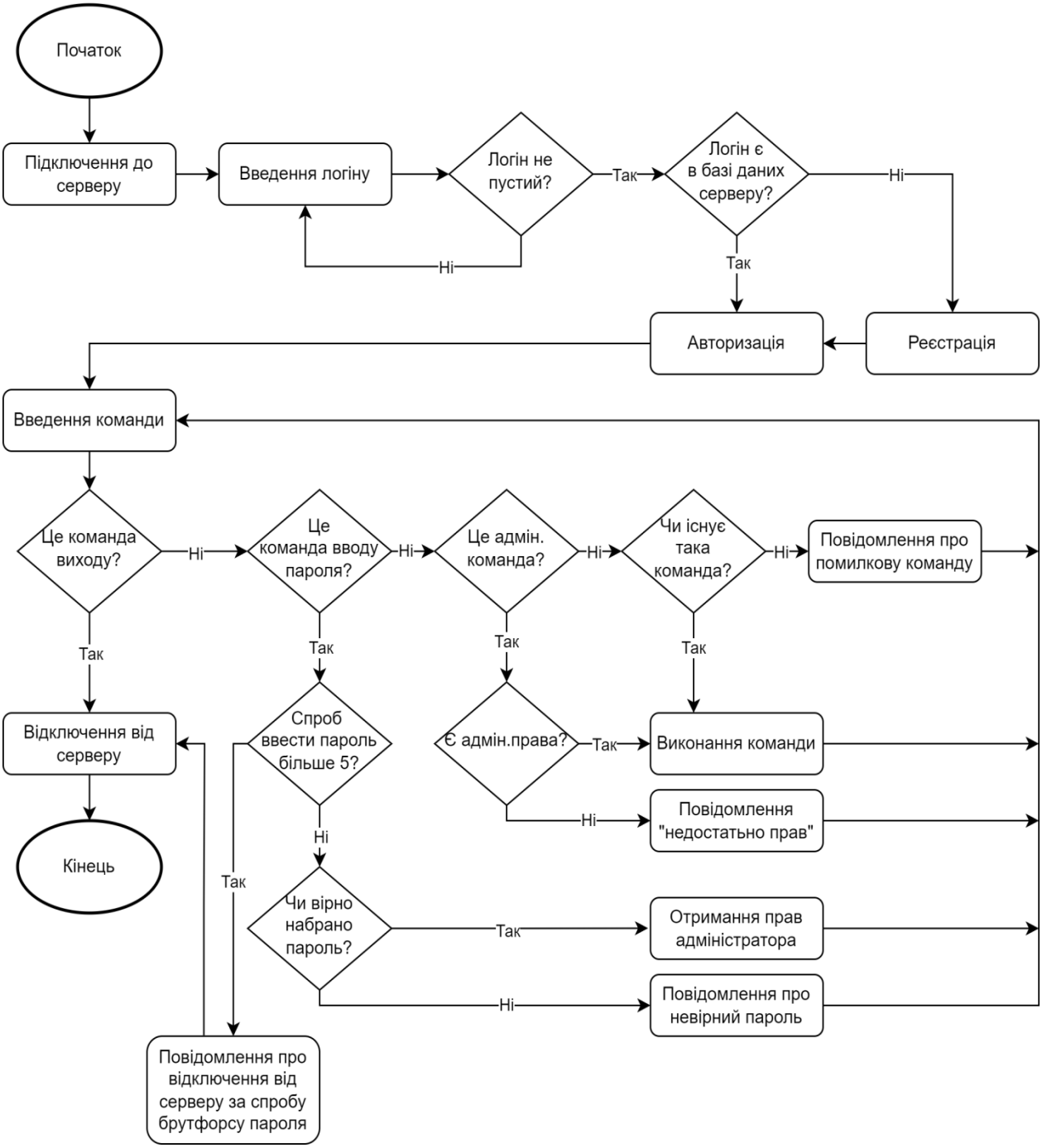

Рисунок 3.2 – Схема функціонування програмного забезпечення для передачі повідомлень у реальному часі

Після входу програмного забезпечення для передачі повідомлень у реальному часі переходить до обробки запитів, що надходять від користувача у вигляді текстових команд. Кожен запит спочатку перевіряється на допустимість, після чого визначається, чи має користувач належні повноваження для його виконання. У разі відсутності відповідних прав або у

випадку помилок у форматі команди програмне забезпечення повідомляє про це та не проводить небажаних операцій. Такий механізм гарантує стабільність роботи та захищає від некоректних дій.

У системі передбачено окремий процес, присвячений роботі з адміністративними можливостями. Для отримання розширених прав користувач вводить спеціальну команду, після чого програма визначає, чи стосується вона переходу до адміністративного режиму. Додатково здійснюється контроль кількості спроб введення пароля, що дає змогу запобігати несанкціонованому доступу. Якщо кількість невдалих спроб перевищує дозволений поріг, сеанс примусово завершується, а підключення припиняється. Якщо ж пароль внесено правильно в межах допустимої кількості спроб, відкривається доступ до адміністративного інтерфейсу.

Основна частина функціонування системи пов’язана з обміном повідомленнями. Після успішної авторизації користувач може надсилати текстові повідомлення, доповнювати їх візуальними елементами або графічними файлами, а система забезпечує доставлення цих даних у режимі максимальної швидкодії. Повідомлення проходять через механізми перевірки, маршрутизації та формування пакета, після чого надсилаються адресатові. Програмне забезпечення реагує на кожен запит у реальному часі, тому передавання інформації відбувається практично миттєво.

У свою чергу адміністратор отримує можливість переглядати активність користувачів та стан системних журналів, а також очищувати їх за потреби. Це дозволяє контролювати роботу сервісу, своєчасно реагувати на нетипову активність і підтримувати ефективність роботи бази даних.

Усі ці процеси працюють узгоджено, створюючи плавний цикл функціонування, у якому автентифікація, перевірка команд, обробка адміністративних запитів та передавання повідомлень формують єдину систему, що забезпечує швидку, безпечну та безперебійну комунікацію в режимі реального часу.

3.3 Розробка бази даних програмного забезпечення для передачі повідомлень у реальному часі

База даних програмного забезпечення для передачі повідомлень у реальному часі організована таким чином, щоб забезпечувати ефективне зберігання та швидкий доступ до інформації про користувачів, їхній стан, а також історію обміну повідомленнями. Логічна модель бази даних відображає ключові сутності, які взаємодіють між собою для підтримки функціонування системи. Основними компонентами є таблиці, що відповідають за користувачів, повідомлення та стан блокування, а також допоміжні структури для списків зареєстрованих учасників і збереження історії обміну даними.

Схему бази даних програмного забезпечення для передачі повідомлень у реальному часі наведено на рис. 3.3.

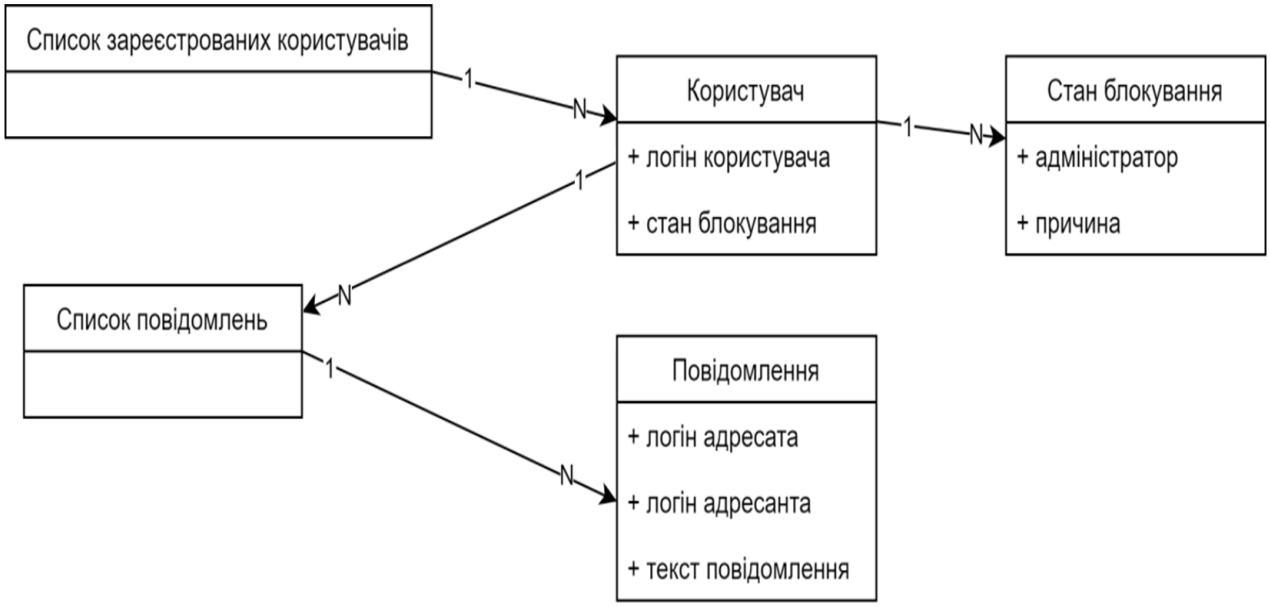

Рисунок 3.3 – Схема бази даних програмного забезпечення для передачі повідомлень у реальному часі

Сутність, що зберігає інформацію про користувачів, включає поля для імені та стану блокування. Ім’я користувача зберігається як рядок, тоді як стан блокування є складовим об’єктом, що містить відомості про адміністратора,

який наклав блокування, та причину блокування. Дані про користувачів формують динамічний масив, який зберігається у зовнішньому текстовому файлі. Така організація дозволяє відокремлювати логіку управління обліковими записами від оперативної пам’яті програми, забезпечуючи одночасно швидкий доступ та довготривале зберігання.

Сутність стану блокування функціонує як вкладений об’єкт у профілі користувача і дає змогу контролювати доступ до системи. Вона реєструє адміністратора, який здійснив блокування, а також текстовий опис причини, що дозволяє відстежувати дії керівника системи і вести журнал потенційних порушень. Така конструкція забезпечує тісний зв’язок між користувачем та його правами, а також дозволяє оперативно змінювати статус у разі відновлення доступу.

Інша сутність, що зберігає повідомлення, включає поля відправника, отримувача та сам текст повідомлення. Ця структура організована як динамічний масив у пам’яті, що забезпечує швидке додавання нових повідомлень і миттєвий доступ для відображення їх користувачам. Зв’язок між таблицями користувачів і повідомлень реалізується через поля, що ідентифікують відправника та отримувача, створюючи відносини «один до багатьох», коли один користувач може відправляти або отримувати багато повідомлень.

Система передбачає наявність допоміжних списків, які формують колекції зареєстрованих користувачів та активних повідомлень. Вони дозволяють ефективно проводити пошук учасників і керувати історією комунікацій, а також забезпечують взаємодію між різними компонентами програми без необхідності звертатися до зовнішніх ресурсів щоразу. Така організація підвищує продуктивність системи та зменшує час доступу до критичних даних.

У логічній моделі всі сутності взаємопов’язані таким чином, щоб забезпечувати цілісність даних та підтримувати коректність роботи системи. Користувачі пов’язані з повідомленнями через поля відправника та

отримувача, а стан блокування інтегрований у профіль користувача, що дозволяє централізовано контролювати права доступу. Ця структура забезпечує швидкий обмін інформацією, стабільність роботи системи та можливість масштабування при збільшенні кількості користувачів і повідомлень.

3.4 Проєктування інтерфейсу програмного забезпечення для передачі повідомлень у реальному часі

Проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для передачі повідомлень у реальному часі виконано з урахуванням вимог технічного завдання. При розробленні програмного забезпечення для передачі повідомлень у реальному часі враховані функціональні вимоги до програми:

– підтримка можливості авторизації до системи передачі повідомлень;

– підтримка можливості відображення списку користувачів, доступних до спілкування;

– можливість виведення списку текстових повідомлень, що відображають процес спілкування;

– можливість відправлення та отримання повідомлень для спілкування з іншими користувачами;

– підтримка можливості використання спеціальних графічних символів (смайли) у процесі спілкування;

– підтримка можливості зміни кольорової гами в інтерфейсі користувача.

Інтерфейс головної форми програмного забезпечення для передачі повідомлень у реальному часі наведено на рис. 3.4.

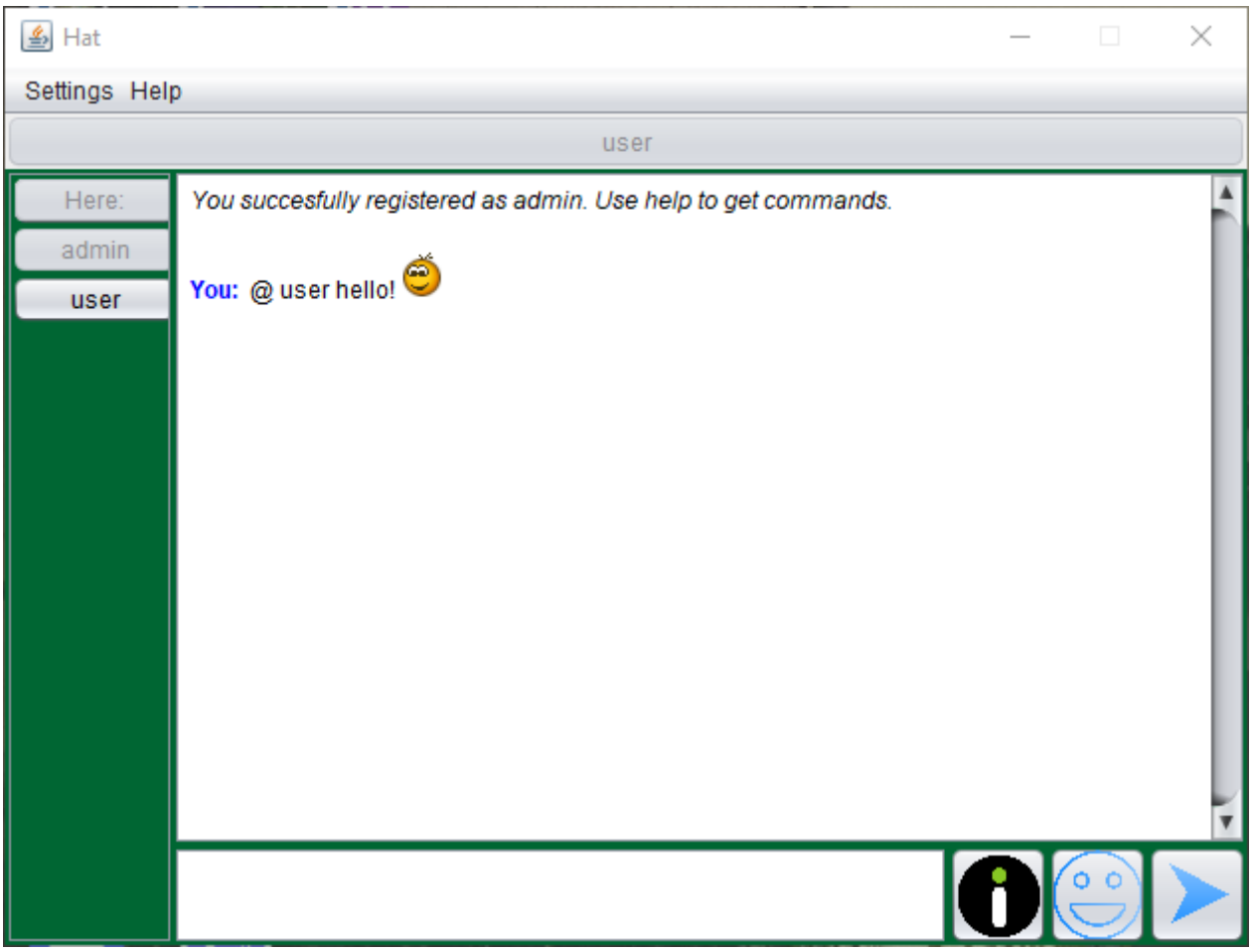

Рисунок 3.4 – Інтерфейс головної форми програмного забезпечення для передачі повідомлень у реальному часі

Інтерфейс форми входу у систему наведено на рис. 3.5.

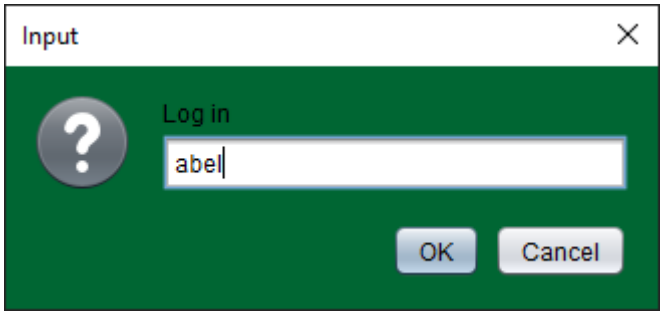

Рисунок 3.5 – Інтерфейс форми входу у систему

Сторінку з прикладом спілкування за допомогою програми наведено на рис. 3.6.

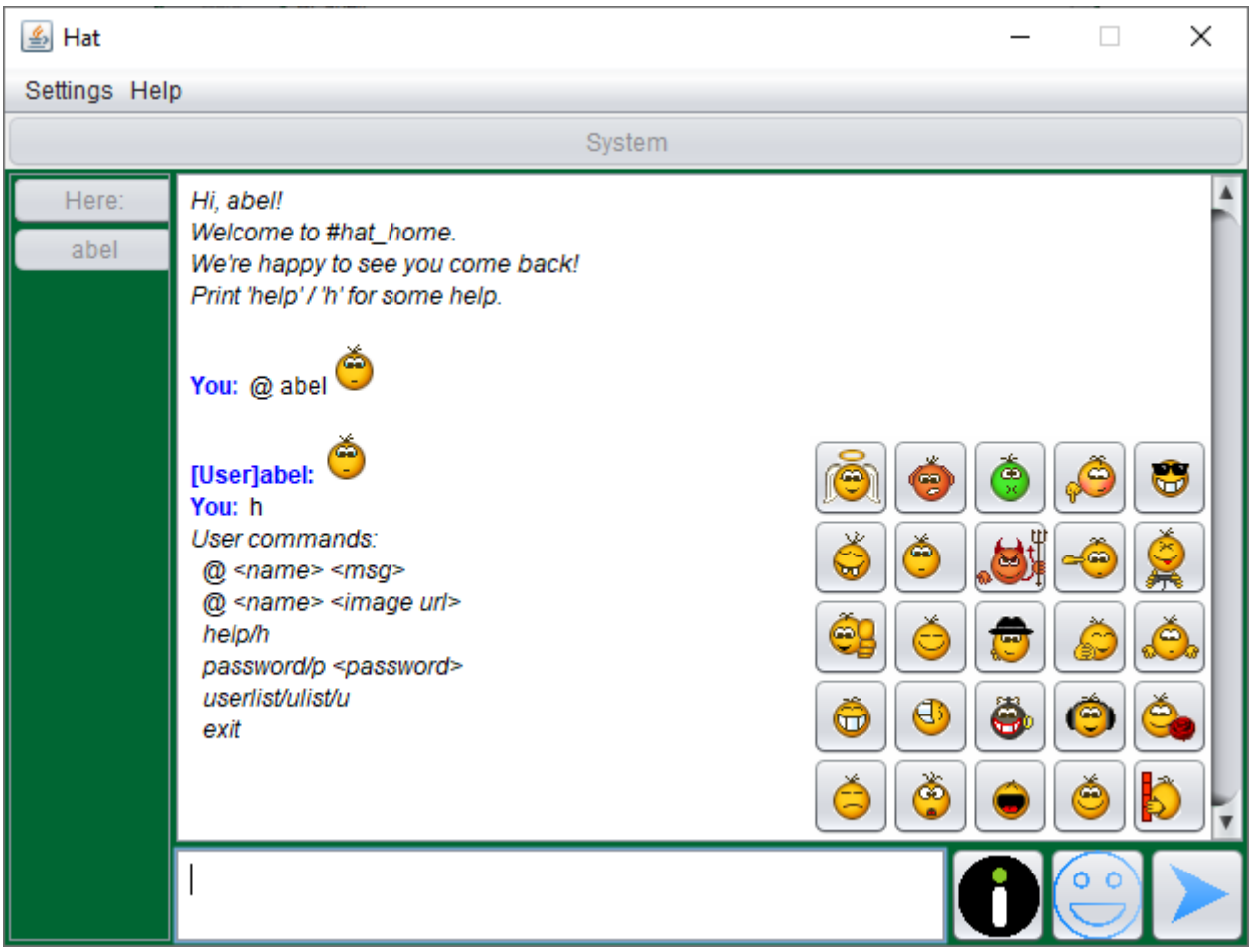

Рисунок 3.6 – Сторінка з прикладом спілкування за допомогою програми

3.5 Висновки за розділом 3

Розроблено програмне забезпечення для передачі повідомлень у реальному часі.

Запропоновано структуру програмного забезпечення для передачі повідомлень у реальному часі. Визначено, що архітектура програмного забезпечення, призначеного для передавання повідомлень у реальному часі, передбачає поділ на кілька взаємопов’язаних модулів, кожен з яких відповідає за окремий аспект роботи системи. Основними модулями є: модуль керування обліковими записами, модуль підтримки роботи адміністратора, модуль обміну текстовими повідомленнями, модуль мультимедійних елементів, модуль журналювання системних подій, модуль взаємодії з мережею.

Описано функціонування програмного забезпечення для передачі повідомлень у реальному часі. Функціонування програмного забезпечення для передачі повідомлень у реальному часі ґрунтується на послідовній взаємодії кількох механізмів, що забезпечують цілісність роботи системи та безперервність комунікації між користувачами. Основна частина функціонування системи пов’язана з обміном повідомленнями. Після успішної авторизації користувач може надсилати текстові повідомлення, а система забезпечує доставлення цих даних у режимі максимальної швидкодії. Повідомлення проходять через механізми перевірки, маршрутизації та формування пакета, після чого надсилаються адресатові. Програмне забезпечення реагує на кожен запит у реальному часі, тому передавання інформації відбувається практично миттєво.

Розроблено базу даних програмного забезпечення для передачі повідомлень у реальному часі, яка організована таким чином, щоб забезпечувати ефективне зберігання та швидкий доступ до інформації про користувачів, їхній стан, а також історію обміну повідомленнями. Логічна модель бази даних відображає ключові сутності, які взаємодіють між собою для підтримки функціонування системи. Основними компонентами є таблиці, що відповідають за користувачів, повідомлення та стан блокування, а також допоміжні структури для списків зареєстрованих учасників і збереження історії обміну даними.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для передачі повідомлень у реальному часі.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для передачі повідомлень у реальному часі. Програма написана на мові Java, яка поєднує стабільність, безпеку та багаторічний досвід використання в критично важливих програмних рішеннях. В якості середовища розробки для створення програмного забезпечення для передачі повідомлень у реальному часі обрано Eclipse, що поєднує структурованість, підтримку сучасних інструментів Java, розширюваність та аналітичні можливості, необхідні для створення систем передачі повідомлень у реальному часі.

Програмне забезпечення для передачі повідомлень у реальному часі забезпечує виконання таких функцій:

– підтримка можливості авторизації до системи передачі повідомлень;

– підтримка можливості відображення списку користувачів, доступних до спілкування;

– можливість виведення списку текстових повідомлень, що відображають процес спілкування;

– можливість відправлення та отримання повідомлень для спілкування з іншими користувачами;

– підтримка можливості використання спеціальних графічних символів (смайли) у процесі спілкування;

– підтримка можливості зміни кольорової гами в інтерфейсі користувача.

Для роботи програмного забезпечення для передачі повідомлень у реальному часі потрібні такі технічні та програмні ресурси: процесор із тактовою частотою не нижче 2,5 ГГц, а також оперативна пам’ять обсягом не менше 8 ГБ, що дозволяє ефективно зберігати тимчасові дані користувачів, масиви повідомлень і забезпечує швидкий доступ до них. Для коректного

функціонування системи необхідний жорсткий диск або твердотільний накопичувач із мінімальним обсягом 256 ГБ для зберігання бази даних користувачів, логів адміністратора та мультимедійних файлів, що передаються у складі повідомлень.

Серед програмних ресурсів важливим є встановлений Java Runtime Environment (JRE) або Java Development Kit (JDK), що забезпечує сумісність із застосованими бібліотеками та дозволяє виконувати багатопоточні операції, необхідні для обробки миттєвих повідомлень. Для серверної частини рекомендується операційна система на базі Linux або Windows із підтримкою мережевих протоколів TCP/IP та сучасного стеку безпеки. Необхідним є наявність встановлених драйверів для підключення до бази даних та підтримка протоколів шифрування для безпечного обміну повідомленнями.

Також система потребує стабільного Інтернет з’єднання з пропускною здатністю не менше 10 Мбіт/с, що гарантує швидку доставку повідомлень користувачам без значних затримок.

Функціональні характеристики розробленого програмного забезпечення для передачі повідомлень у реальному часі наведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для передачі повідомлень у реальному часі наведено у додатку Б.

4.2 Характеристики програми для передачі повідомлень у реальному часі

Програмне забезпечення для передачі повідомлень у реальному часі характеризується високою швидкістю обробки даних та здатністю забезпечувати миттєву комунікацію між користувачами незалежно від кількості одночасно підключених учасників. Воно відзначається стійкістю до навантажень та здатністю підтримувати стабільну роботу навіть при великій кількості активних сеансів, що дозволяє користувачам взаємодіяти без затримок і перебоїв. Система проявляє гнучкість у розширенні функціоналу,

що дає змогу додавати нові засоби обміну інформацією та інтегрувати мультимедійні елементи без втрати продуктивності.

Також програмне забезпечення для передачі повідомлень у реальному часі вирізняється високим рівнем безпеки, забезпечуючи контроль доступу, захист даних користувачів та механізми запобігання несанкціонованим спробам доступу. Його архітектура орієнтована на модульність, що дозволяє ізолювати різні компоненти системи, знижуючи ризики виникнення помилок та спрощуючи процес обслуговування. Взаємодія між модулями відбувається через чітко визначені інтерфейси, що підвищує надійність та передбачуваність роботи системи.

Програмне забезпечення для передачі повідомлень у реальному часі демонструє адаптивність до різних платформ та операційних середовищ, забезпечуючи користувачам однаково комфортний досвід незалежно від типу пристрою чи конфігурації системи. Його функціональні можливості включають як обробку текстових повідомлень, так і передавання графічних елементів, що підвищує інформативність та емоційну насиченість спілкування. Всі ці характеристики забезпечують ефективну роботу програми та підтримують безперервну комунікацію в режимі реального часу, зменшуючи затрати ресурсів та підвищуючи продуктивність взаємодії користувачів.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для запуску програмного забезпечення для передачі повідомлень у реальному часі на серверній частини необхідно запустити відповідний java-файл. Для звертання до клієнтської сторони необхідно запустити клієнтську частину та ввести дані користувача.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програмного забезпечення для передачі повідомлень у реальному часі є інформація, яку користувач вводить для ідентифікації та доступу до системи, включаючи логін та пароль, а також текстові повідомлення, мультимедійні файли та інші елементи, що супроводжують обмін повідомленнями.

Вихідними даними програмного забезпечення для передачі повідомлень у реальному часі є доставлені адресатам текстові повідомлення та мультимедійні файли, підтвердження авторизації користувачів та інші повідомлення.

4.3.3 Повідомлення

Користувач програмного забезпечення для передачі повідомлень у реальному часі може отримати такі повідомлення: підтвердження успішного входу або помилки при введенні даних, сповіщення про неможливість недоставки повідомлення, повідомлення від інших користувачів, а також системні повідомлення.

4.4 Виконання програмного забезпечення для передачі повідомлень у реальному часі

Після запуску програмного забезпечення для передачі повідомлень у реальному часі користувачу відображається форма входу (рис. 4.1), де потрібно ввести свій логін, після чого ввести пароль.

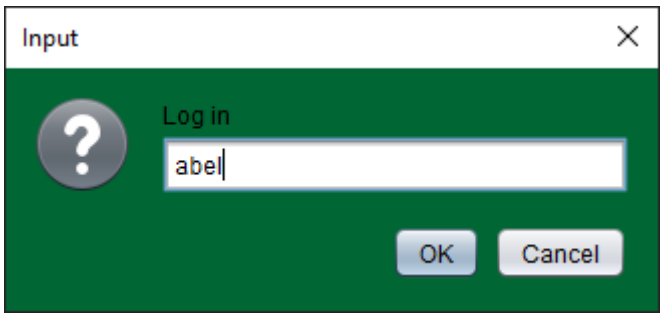

Рисунок 4.1 – Форма входу до програмного забезпечення для передачі повідомлень у реальному часі

Після входу до системи користувач може бачити головну форму (рис. 4.2). У головній формі зверху відображається меню та ім’я користувача, зліва знаходиться панель зі списком користувачів, доступних до спілкування. В основній області розташовано список повідомлень, що відображають процес спілкування користувача.

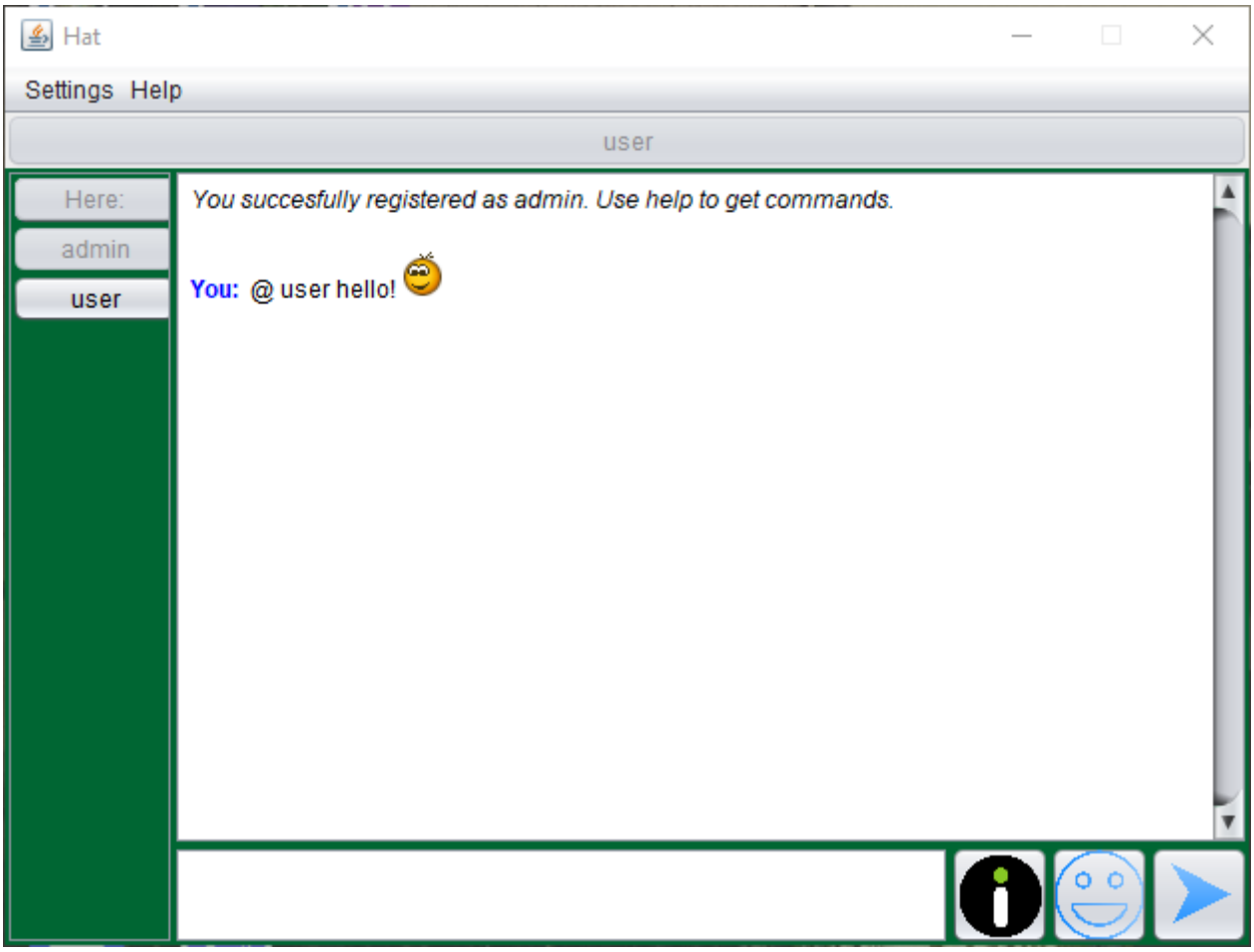

Рисунок 4.2 – Головна форма програми

Для спілкування (рис. 4.3) необхідно обрати зі списку потрібного користувача, ввести текстове повідомлення та натиснути відповідну кнопку для відправлення. При натисканні кнопки зі значком посмішки відображається панель зображень (смайлів), які можна використовувати у процесі спілкування.

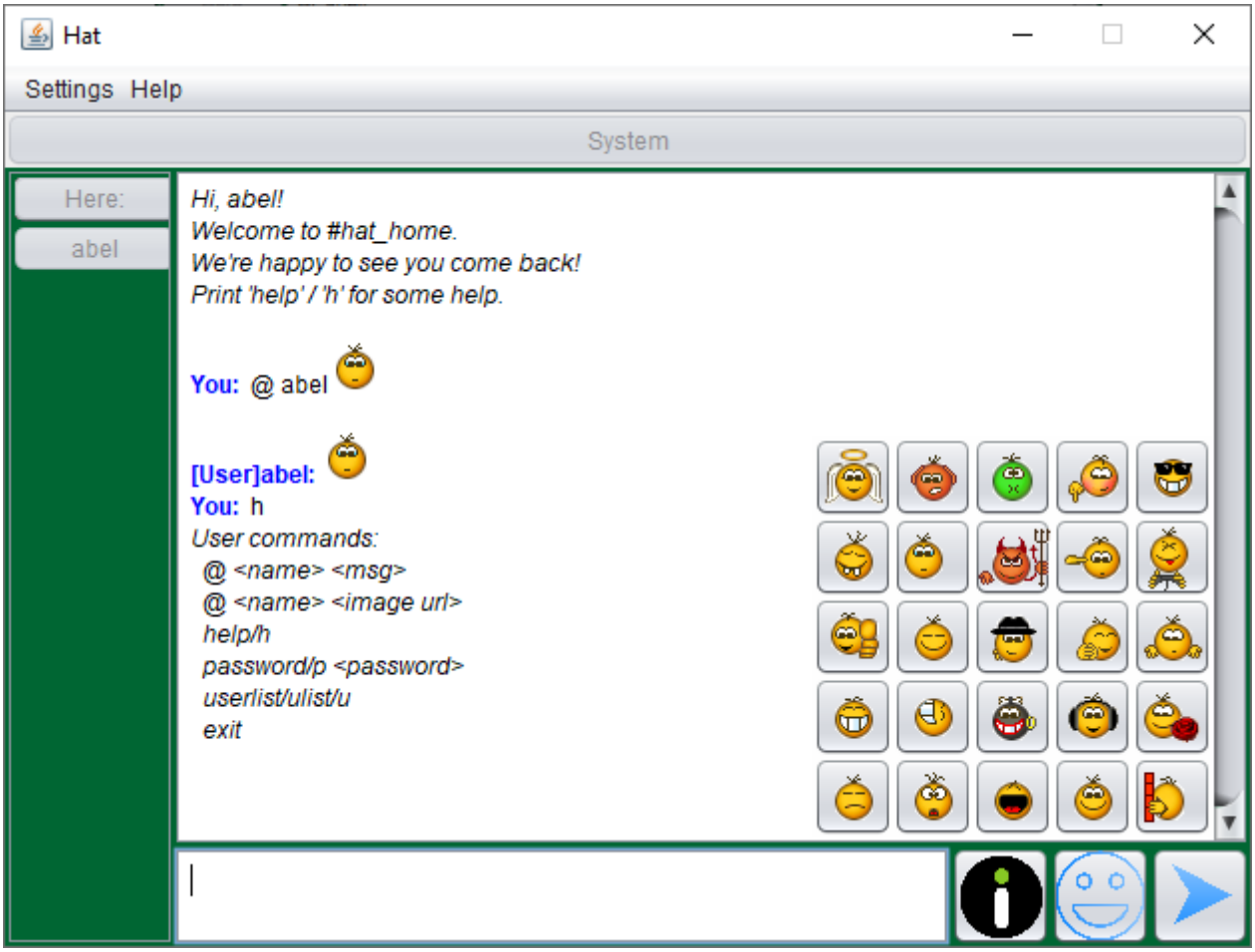

Рисунок 4.3 – Спілкування за допомогою програми

При необхідності зміни кольорової гами в інтерфейсі користувача (рис. 4.4) треба у головному меню обрати пункт Settings та вибрати потрібний колір.

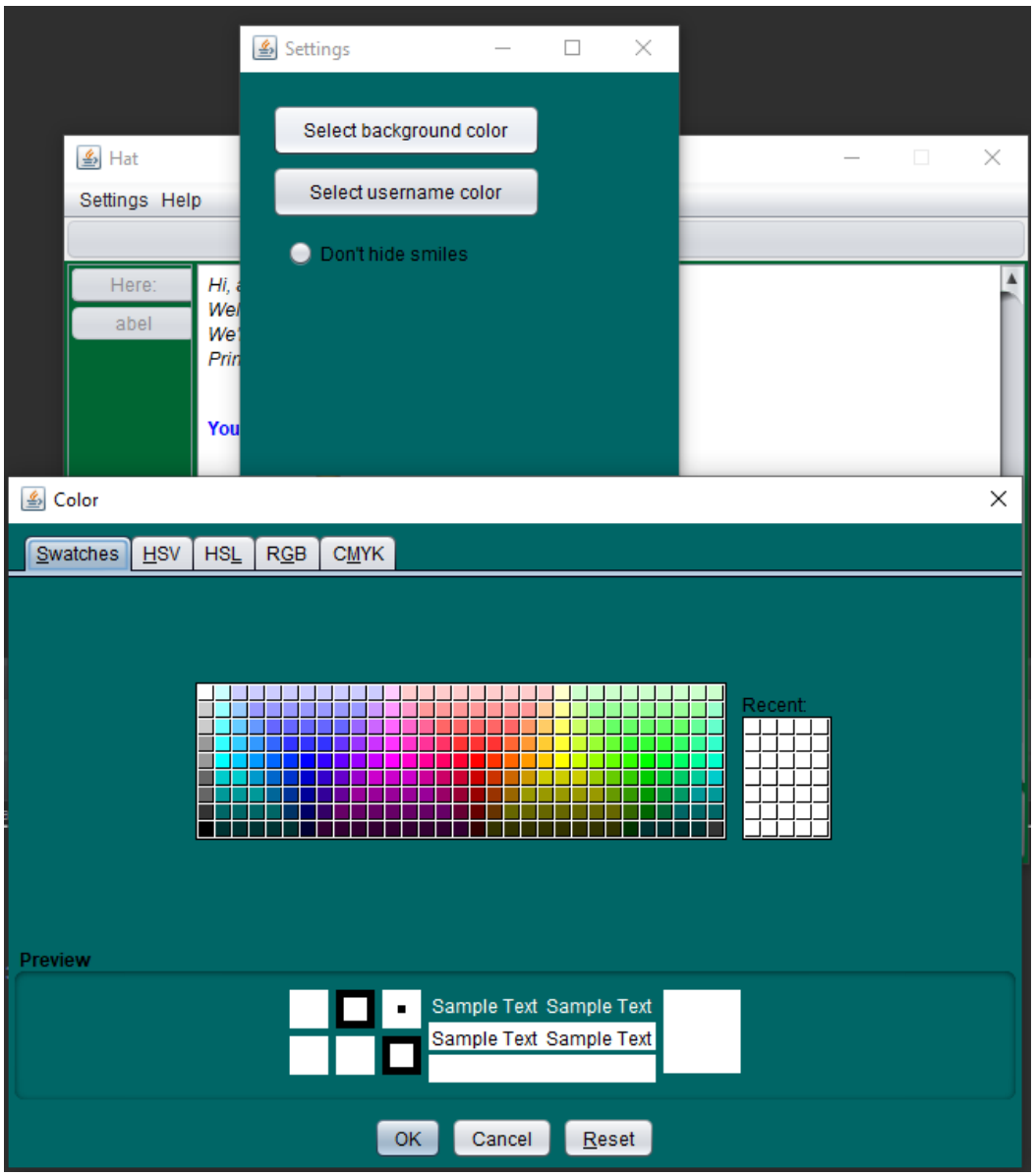

Рисунок 4.4 – Зміна кольорової гами в інтерфейсі користувача

4.5 Тестування програмного забезпечення для передачі повідомлень у реальному часі

Виконано тестування програмного забезпечення для передачі повідомлень у реальному часі.

Виявлено, що програма для передачі повідомлень у реальному часі функціонує правильно та злагоджено. Вона реалізує всі функціональні вимоги та успішно виконує свою основну задачу передачі повідомлень між користувачами у реальному часі.

Розроблене програмне забезпечення дозволяє забезпечувати автоматизацію процесів, пов’язаних з підтримкою процесу передачі повідомлень між користувачами у реальному часі.

4.6 Висновки за розділом 4

Описано програмне забезпечення для передачі повідомлень у реальному часі. Виконано тестування розробленого програмного забезпечення для передачі повідомлень у реальному часі. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов’язаних з підтримкою процесу передачі повідомлень між користувачами у реальному часі.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процес розробки програмного забезпечення для передачі повідомлень у реальному часі.

Визначено, що програмні засоби для передачі повідомлень у реальному часі забезпечують миттєву взаємодію між користувачами та дозволяють інформації надходити без затримок. Таке програмне забезпечення здатне обробляти повідомлення відразу після їхнього створення, що робить спілкування максимально природним і наближеним до живого діалогу. Подібні програмні рішення створюються для того, щоб підтримувати постійний потік даних між учасниками комунікації, незалежно від їхнього місцезнаходження чи типу пристрою.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для передачі повідомлень у реальному часі. Проте деякі програмні засоби для передачі повідомлень у реальному часі можуть не забезпечувати стабільну роботу в умовах високого навантаження. Проблемою залишається і забезпечення безпеки. Передавання даних у реальному часі потребує постійного шифрування та захисту каналів зв’язку, а також контролю за цілісністю даних у момент їх передачі. У багатьох випадках користувачі змушені довіряти стороннім провайдерам критично важливу інформацію, що не завжди відповідає їхнім вимогам до приватності. Тому актуальною є розробка програмного забезпечення для передачі повідомлень у реальному часі.

Сформульовано функціональні вимоги до програмного забезпечення для передачі повідомлень у реальному часі.

Для реалізації програмного забезпечення для передачі повідомлень у реальному часі обрано мову програмування Java, яка поєднує стабільність, безпеку та багаторічний досвід використання в критично важливих програмних рішеннях. Крім того, екосистема Java пропонує широкий набір

програмних рішень для побудови високонавантажених серверних платформ, де підтримка потоків, асинхронних операцій і неблокувальних механізмів дозволяє ефективно розподіляти обчислювальну роботу. Це створює передумови для побудови програмних сервісів, які залишаються чутливими навіть при значному навантаженні.

Для створення програмного забезпечення для передачі повідомлень у реальному часі обрано середовище розробки Eclipse, що поєднує структурованість, підтримку сучасних інструментів Java, розширюваність та аналітичні можливості, необхідні для створення систем передачі повідомлень у реальному часі. Його використання забезпечує контрольованість, ефективність та передбачуваність усіх етапів розробки.

Розроблено програмне забезпечення для передачі повідомлень у реальному часі.

Запропоновано структуру програмного забезпечення для передачі повідомлень у реальному часі. Визначено, що архітектура програмного забезпечення, призначеного для передавання повідомлень у реальному часі, передбачає поділ на кілька взаємопов’язаних модулів, кожен з яких відповідає за окремий аспект роботи системи. Основними модулями є: модуль керування обліковими записами, модуль підтримки роботи адміністратора, модуль обміну текстовими повідомленнями, модуль мультимедійних елементів, модуль журналювання системних подій, модуль взаємодії з мережею.

Описано функціонування програмного забезпечення для передачі повідомлень у реальному часі. Функціонування програмного забезпечення для передачі повідомлень у реальному часі ґрунтується на послідовній взаємодії кількох механізмів, що забезпечують цілісність роботи системи та безперервність комунікації між користувачами. Основна частина функціонування системи пов’язана з обміном повідомленнями. Після успішної авторизації користувач може надсилати текстові повідомлення, а система забезпечує доставлення цих даних у режимі максимальної швидкодії. Повідомлення проходять через механізми перевірки, маршрутизації та

формування пакета, після чого надсилаються адресатові. Програмне забезпечення реагує на кожен запит у реальному часі, тому передавання інформації відбувається практично миттєво.

Розроблено базу даних програмного забезпечення для передачі повідомлень у реальному часі, яка організована таким чином, щоб забезпечувати ефективне зберігання та швидкий доступ до інформації про користувачів, їхній стан, а також історію обміну повідомленнями. Логічна модель бази даних відображає ключові сутності, які взаємодіють між собою для підтримки функціонування системи. Основними компонентами є таблиці, що відповідають за користувачів, повідомлення та стан блокування, а також допоміжні структури для списків зареєстрованих учасників і збереження історії обміну даними.

Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для передачі повідомлень у реальному часі.

Виконано тестування розробленого програмного забезпечення для передачі повідомлень у реальному часі. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов’язаних з підтримкою процесу передачі повідомлень між користувачами у реальному часі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is Real-Time Messaging and What are the Best Platforms. URL: https://www.podium.com/article/real-time-messaging-2 (date of access: 25.04.2026).

2. What are Real Time Messaging Apps. URL: https://www.pubnub.com/guides/real-time-messaging-benefits-basics-and-first-steps/ (date of access: 25.04.2026).

3. Understanding Real-Time Messaging Protocol: How It Powers Instant Communication. URL: https://ipwithease.com/real-time-messaging-protocol/ (date of access: 25.04.2026).

4. Stay connected in real time with the right messaging app. URL: https://www.insideonemagazine.com/stay-connected-in-real-time-with-the-right-messaging-app/ (date of access: 25.04.2026).

5. Real-Time Chat. URL: https://getstream.io/glossary/real-time-chat/ (date of access: 25.04.2026).

6. Complete Guide to RTMP: The Real-Time Messaging Protocol. URL: https://www.wowza.com/blog/rtmp (date of access: 25.04.2026).

7. Instant Messaging Guide: Definition, How It Works & Examples. URL: https://www.zegocloud.com/blog/instant-messaging (date of access: 25.04.2026).

8. Real-Time Communication Tools That Make Remote Work Feel Human Again. URL: https://www.chanty.com/blog/real-time-communication-tools/ (date of access: 25.04.2026).

9. Best Live Chat Software Shortlist. URL: https://thecxlead.com/tools/best-live-chat-software/ (date of access: 25.04.2026).

10. Best Real-time Language Translation Apps for Effortless Communication. URL: https://www.aiphone.ai/blog/best-real-time-translation-apps/ (date of access: 25.04.2026).

11. Instant messaging software. URL: https://www.zendesk.es/service/messaging/instant-messaging-software/ (date of access: 25.04.2026).

12. Open-source Chat Servers For Building Real-time Chat Experiences for Games, and Social Apps. URL: https://medevel.com/26-os-chat-servers/ (date of access: 25.04.2026).

13. Connecteam. URL: https://connecteam.com/ (date of access: 25.04.2026).

14. Quidget. URL: https://quidget.ai/live-chat/ (date of access: 25.04.2026).

15. MirrorFly. URL: https://www.mirrorfly.com/ (date of access: 25.04.2026).

16. The Destination for Java Developers. URL: https://dev.java/ (date of access: 25.04.2026).

17. Java Annotated Monthly. URL: https://blog.jetbrains.com/idea/2025/12/java-annotated-monthly-december-2025/ (date of access: 25.04.2026).

18. Eclipse IDE Overview: How to Install, Pros & Cons, Price. URL: https://www.omi.me/blogs/overview/eclipse-ide-overview-how-to-install-pros-cons-price (date of access: 25.04.2026).

19. Eclipse documentation. URL: https://help.eclipse.org/2025-09/index.jsp (date of access: 25.04.2026).

ДОДАТОК А

Технічне завдання

Вступ

Програмне забезпечення може використовуватися для передачі повідомлень у реальному часі.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для передачі повідомлень у реальному часі», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 7 квітня 2026 р.

А.2 Призначення розробки

Програмний продукт призначений для передачі повідомлень у реальному часі.

А.3 Основні вимоги до програми, що розробляється

А.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для передачі повідомлень у реальному часі забезпечує виконання таких функцій:

– підтримка можливості авторизації до системи передачі повідомлень;

– підтримка можливості відображення списку користувачів, доступних до спілкування;

– можливість виведення списку текстових повідомлень, що відображають процес спілкування;

– можливість відправлення та отримання повідомлень для спілкування з іншими користувачами;

– підтримка можливості використання спеціальних графічних символів (смайли) у процесі спілкування;

– підтримка можливості зміни кольорової гами в інтерфейсі користувача.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для передачі повідомлень у реальному часі повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному режимі.

А.3.3 Вимоги до надійності

Програмне забезпечення для передачі повідомлень у реальному часі повинно забезпечити надійне функціонування.

А.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для передачі повідомлень у реальному часі необхідна наявність персонального комп’ютера.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для роботи програмного забезпечення для передачі повідомлень у реальному часі потрібні такі технічні та програмні ресурси: процесор із тактовою частотою не нижче 2,5 ГГц, а також оперативна пам’ять обсягом не менше 8 ГБ, що дозволяє ефективно зберігати тимчасові дані користувачів, масиви повідомлень і забезпечує швидкий доступ до них. Для коректного

функціонування системи необхідний жорсткий диск або твердотільний накопичувач із мінімальним обсягом 256 ГБ для зберігання бази даних користувачів, логів адміністратора та мультимедійних файлів, що передаються у складі повідомлень.

Серед програмних ресурсів важливим є встановлений Java Runtime Environment або Java Development Kit, що забезпечує сумісність із застосованими бібліотеками та дозволяє виконувати багатопоточні операції, необхідні для обробки миттєвих повідомлень. Для серверної частини рекомендується операційна система на базі Linux або Windows із підтримкою мережевих протоколів TCP/IP та сучасного стеку безпеки. Необхідним є наявність встановлених драйверів для підключення до бази даних та підтримка протоколів шифрування для безпечного обміну повідомленнями.

Також система потребує стабільного Інтернет з’єднання з пропускною здатністю не менше 10 Мбіт/с, що гарантує швидку доставку повідомлень користувачам без значних затримок.

А.3.6 Вимоги до маркування і пакування

Програма для передачі повідомлень у реальному часі може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для передачі повідомлень у реальному часі».

А.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення для передачі повідомлень у реальному часі є інформація, яку користувач вводить для ідентифікації та доступу до системи, включаючи логін та пароль, а також текстові повідомлення, мультимедійні файли та інші елементи, що супроводжують обмін повідомленнями.

Вихідними даними програмного забезпечення для передачі повідомлень у реальному часі є доставлені адресатам текстові повідомлення та мультимедійні файли, підтвердження авторизації користувачів та інші повідомлення.

ДОДАТОК Б

Фрагмент тексту програми

```
public class Srv {
    private ServerSocket svrSkt;
    private ExecutorService thrdPl;
    private Map<String, CltHndl> pplHndl;
    private volatile boolean runFlg;
    private int prt;

    public Srv(int prtN) {
        prt = prtN;
        pplHndl = new ConcurrentHashMap<>();
        thrdPl = Executors.newCachedThreadPool();
    }

    public void strt() throws IOException {
        svrSkt = new ServerSocket(prt);
        runFlg = true;
        thrdPl.submit(() -> {
            while (runFlg) {
                try {
                    Socket sck = svrSkt.accept();
                    thrdPl.submit(new CltHndl(sck));
                } catch (IOException ex) {
                    if (!runFlg) break;
                }
            }
        });
    }

    public void stp() throws IOException {
        runFlg = false;
        if (svrSkt != null) svrSkt.close();
        thrdPl.shutdownNow();
        for (CltHndl h : pplHndl.values()) {
            h.clse();
        }
        pplHndl.clear();
    }

    private void sndMsgAll(String hdr, byte[] pld) {
        for (CltHndl h : pplHndl.values()) {
            h.sndRaw(hdr, pld);
        }
    }

    private void sndMsgTo(String tgt, String hdr, byte[] pld) {
        CltHndl h = pplHndl.get(tgt);
        if (h != null) h.sndRaw(hdr, pld);
```

```
    }

    private void brdPplLst() {
        StringBuilder sb = new StringBuilder();
        sb.append("PPL|");
        boolean fst = true;
        for (String nm : pplHndl.keySet()) {
            if (!fst) sb.append(",");
            sb.append(nm);
            fst = false;
        }
        byte[] pld = sb.toString().getBytes();
        sndMsgAll("SYS|PPLUPD|" + pld.length, pld);
    }

    private class CltHndl implements Runnable {
        private Socket sck;
        private DataInputStream din;
        private DataOutputStream dout;
        private String nm;
        private volatile boolean act;

        public CltHndl(Socket s) {
            sck = s;
            try {
                sck.setSoTimeout(0);
                din = new DataInputStream(new
BufferedInputStream(sck.getInputStream()));
                dout = new DataOutputStream(new
BufferedOutputStream(sck.getOutputStream()));
                act = true;
                thrdPl.submit(this);
            } catch (IOException ex) {
                act = false;
                clse();
            }
        }

        public void run() {
            try {
                while (act) {
                    String hdr = rdHdr();
                    if (hdr == null) break;
                    String[] toks = hdr.split("\\|");
                    if (toks.length < 3) continue;
                    if ("CMD".equals(toks[0])) {
                        handleCmd(toks);
                    } else if ("MSG".equals(toks[0])) {
```

```
                        int len =
Integer.parseInt(toks[toks.length - 1]);
                        byte[] pld = rdPld(len);
                        if (pld == null) continue;
                        String tp = toks[2];
                        String snd = toks[1];
                        if ("ALL".equals(tp)) {
                            sndMsgAll(hdr, pld);
                        } else {
                            String tgt = tp;
                            sndMsgTo(tgt, hdr, pld);
                        }
                    } else if ("BIN".equals(toks[0])) {
                        int len =
Integer.parseInt(toks[toks.length - 1]);
                        byte[] pld = rdPld(len);
                        if (pld == null) continue;
                        String snd = toks[1];
                        String tgt = toks[2];
                        String fnm = toks[3];
                        sndMsgTo(tgt, hdr, pld);
                    }
                }
            } catch (Exception ex) {
                // ignore
            } finally {
                clse();
            }
        }

        private String rdHdr() {
            try {
                String ln = din.readUTF();
                return ln;
            } catch (IOException ex) {
                act = false;
                return null;
            }
        }

        private byte[] rdPld(int len) {
            try {
                byte[] b = new byte[len];
                din.readFully(b);
                return b;
            } catch (IOException ex) {
                act = false;
                return null;
```

```
            }
        }

        private void handleCmd(String[] toks) {
            if (toks.length < 2) return;
            String cmd = toks[1];
            if ("LOGIN".equals(cmd)) {
                if (toks.length < 3) return;
                String nmIn = toks[2];
                if (nmIn == null || nmIn.trim().isEmpty())
return;
                nm = nmIn.trim();
                pplHndl.put(nm, this);
                brdPplLst();
            } else if ("LOGOUT".equals(cmd)) {
                clse();
            }
        }

        public synchronized void sndRaw(String hdr, byte[] pld) {
            try {
                dout.writeUTF(hdr);
                if (pld != null && pld.length > 0)
dout.write(pld);
                dout.flush();
            } catch (IOException ex) {
                clse();
            }
        }

        public synchronized void clse() {
            act = false;
            if (nm != null) {
                pplHndl.remove(nm);
                brdPplLst();
            }
            try {
                if (din != null) din.close();
            } catch (IOException e) {}
            try {
                if (dout != null) dout.close();
            } catch (IOException e) {}
            try {
                if (sck != null) sck.close();
            } catch (IOException e) {}
        }
    }
```

```
    public static void main(String[] args) {
        int p = 9000;
        Srv srv = new Srv(p);
        try {
            srv.strt();
            System.out.println("Srv strtd prt=" + p);
        } catch (IOException e) {
            System.out.println("Srv err");
        }
    }
}

class Clt {
    private String nm;
    private Socket sck;
    private DataInputStream din;
    private DataOutputStream dout;
    private CltWnd wnd;
    private volatile boolean runFlg;
    private ExecutorService thrdPl;

    public Clt() {
        thrdPl = Executors.newCachedThreadPool();
        wnd = new CltWnd(this);
    }

    public void cnct(String hst, int prt, String nmIn) {
        try {
            sck = new Socket(hst, prt);
            din = new DataInputStream(new
BufferedInputStream(sck.getInputStream()));
            dout = new DataOutputStream(new
BufferedOutputStream(sck.getOutputStream()));
            nm = nmIn;
            sndCmd("LOGIN|" + nm);
            runFlg = true;
            thrdPl.submit(() -> rcvLoop());
            wnd.shw();
        } catch (IOException ex) {
            wnd.ntf("cnct flld: " + ex.getMessage());
        }
    }

    public void dsct() {
        try {
            sndCmd("LOGOUT");
        } catch (Exception e) {}
        runFlg = false;
```

```
        try {
            if (din != null) din.close();
        } catch (IOException e) {}
        try {
            if (dout != null) dout.close();
        } catch (IOException e) {}
        try {
            if (sck != null) sck.close();
        } catch (IOException e) {}
        thrdPl.shutdownNow();
    }

    private void rcvLoop() {
        try {
            while (runFlg) {
                String hdr = din.readUTF();
                if (hdr == null) break;
                String[] toks = hdr.split("\\|");
                if (toks.length < 2) continue;
                if ("SYS".equals(toks[0])) {
                    if ("PPLUPD".equals(toks[1])) {
                        int len = Integer.parseInt(toks[2]);
                        byte[] pld = new byte[len];
                        din.readFully(pld);
                        String s = new String(pld);
                        String[] ppl = s.split(",");
                        wnd.updPplLst(ppl);
                    }
                } else if ("MSG".equals(toks[0])) {
                    String snd = toks[1];
                    String tgt = toks[2];
                    int len = Integer.parseInt(toks[toks.length -
1]);
                    byte[] pld = new byte[len];
                    din.readFully(pld);
                    String txt = new String(pld);
                    if ("ALL".equals(tgt) || nm.equals(tgt) ||
snd.equals(nm)) {
                        wnd.addMsg(snd, txt);
                    }
                } else if ("BIN".equals(toks[0])) {
                    String snd = toks[1];
                    String tgt = toks[2];
                    String fnm = toks[3];
                    int len = Integer.parseInt(toks[4]);
                    byte[] pld = new byte[len];
                    din.readFully(pld);
```

```
                    if (nm.equals(tgt) || "ALL".equals(tgt) ||
snd.equals(nm)) {
                        wnd.addBin(snd, fnm, pld);
                    }
                }
            }
        } catch (IOException ex) {
            wnd.ntf("rcv err: " + ex.getMessage());
        } finally {
            dsct();
        }
    }

    public void sndTxt(String tgt, String txt) {
        try {
            byte[] pld = txt.getBytes();
            String hdr = "MSG|" + nm + "|" + tgt + "|" +
pld.length;
            dout.writeUTF(hdr);
            dout.write(pld);
            dout.flush();
            if (tgt.equals("ALL") || tgt.equals(nm)) {
                wnd.addMsg(nm, txt);
            }
        } catch (IOException ex) {
            wnd.ntf("snd err: " + ex.getMessage());
        }
    }

    public void sndFile(String tgt, File f) {
        try {
            byte[] bin = Files.readAllBytes(f.toPath());
            String fnm = f.getName();
            String hdr = "BIN|" + nm + "|" + tgt + "|" + fnm +
"|" + bin.length;
            dout.writeUTF(hdr);
            dout.write(bin);
            dout.flush();
            wnd.addMsg(nm, "[snd fl] " + fnm);
        } catch (IOException ex) {
            wnd.ntf("snd fl err: " + ex.getMessage());
        }
    }

    private void sndCmd(String cmd) throws IOException {
        String hdr = "CMD|" + cmd;
        if (dout != null) {
            dout.writeUTF(hdr);
```

```
            dout.flush();
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            Clt clt = new Clt();
            clt.wnd.shwLgn();
        });
    }
}

class CltWnd {
    private Clt clt;
    private JFrame frm;
    private JTextPane txtArea;
    private DefaultStyledDocument doc;
    private JTextField txtInp;
    private JButton sndBtn;
    private JList<String> pplLst;
    private DefaultListModel<String> pplMdl;
    private JLabel stLbl;
    private JButton flBtn;
    private JFileChooser flCh;
    private JPanel pnl;

    public CltWnd(Clt c) {
        clt = c;
        frm = new JFrame("CltWnd");
        frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frm.setSize(800, 600);
        frm.setLocationRelativeTo(null);
        pnl = new JPanel(new BorderLayout());
        txtArea = new JTextPane();
        txtArea.setEditable(false);
        doc = (DefaultStyledDocument) txtArea.getDocument();
        JScrollPane sp = new JScrollPane(txtArea);
        pnl.add(sp, BorderLayout.CENTER);
        JPanel rtPnl = nrtPnl();
        pnl.add(rtPnl, BorderLayout.EAST);
        JPanel btPnl = nbtPnl();
        pnl.add(btPnl, BorderLayout.SOUTH);
        frm.getContentPane().add(pnl);
        flCh = new JFileChooser();
        flCh.setFileFilter(new FileNameExtensionFilter("Imgs &
Docs", "jpg", "png", "gif", "pdf", "txt"));
    }
```

```
    private JPanel nrtPnl() {
        JPanel p = new JPanel(new BorderLayout());
        pplMdl = new DefaultListModel<>();
        pplLst = new JList<>(pplMdl);
        JScrollPane sp = new JScrollPane(pplLst);
        p.add(sp, BorderLayout.CENTER);
        stLbl = new JLabel("st: n/a");
        p.add(stLbl, BorderLayout.SOUTH);
        return p;
    }

    private JPanel nbtPnl() {
        JPanel p = new JPanel(new BorderLayout());
        txtInp = new JTextField();
        sndBtn = new JButton("snd");
        flBtn = new JButton("fl");
        JPanel r = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        r.add(flBtn);
        r.add(sndBtn);
        p.add(txtInp, BorderLayout.CENTER);
        p.add(r, BorderLayout.EAST);
        sndBtn.addActionListener(e -> {
            String ms = txtInp.getText();
            if (ms == null || ms.trim().isEmpty()) return;
            String tgt = pplLst.getSelectedValue();
            if (tgt == null || tgt.trim().isEmpty()) tgt = "ALL";
            clt.sndTxt(tgt, ms);
            txtInp.setText("");
        });
        flBtn.addActionListener(e -> {
            int rv = flCh.showOpenDialog(frm);
            if (rv == JFileChooser.APPROVE_OPTION) {
                File f = flCh.getSelectedFile();
                String tgt = pplLst.getSelectedValue();
                if (tgt == null || tgt.trim().isEmpty()) tgt =
"ALL";
                clt.sndFile(tgt, f);
            }
        });
        return p;
    }

    public void shwLgn() {
        JDialog dlg = new JDialog((Frame) null, "lgn", true);
        dlg.setSize(400, 200);
        dlg.setLocationRelativeTo(null);
        JPanel p = new JPanel(new BorderLayout());
        JPanel f = new JPanel(new GridLayout(3, 2));
```

```
        JLabel l1 = new JLabel("hst");
        JTextField hstF = new JTextField("localhost");
        JLabel l2 = new JLabel("prt");
        JTextField prtF = new JTextField("9000");
        JLabel l3 = new JLabel("nm");
        JTextField nmF = new JTextField();
        f.add(l1);
        f.add(hstF);
        f.add(l2);
        f.add(prtF);
        f.add(l3);
        f.add(nmF);
        p.add(f, BorderLayout.CENTER);
        JPanel b = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        JButton cn = new JButton("cnct");
        JButton cl = new JButton("cl");
        b.add(cl);
        b.add(cn);
        p.add(b, BorderLayout.SOUTH);
        dlg.getContentPane().add(p);
        cn.addActionListener(ev -> {
            String h = hstF.getText();
            int pr = Integer.parseInt(prtF.getText());
            String nm = nmF.getText();
            if (nm == null || nm.trim().isEmpty()) {
                JOptionPane.showMessageDialog(dlg, "nm rl");
                return;
            }
            dlg.dispose();
            clt.cnct(h, pr, nm);
        });
        cl.addActionListener(ev -> {
            dlg.dispose();
            System.exit(0);
        });
        dlg.setVisible(true);
    }

    public void shw() {
        SwingUtilities.invokeLater(() -> {
            frm.setVisible(true);
        });
    }

    public void addMsg(String snd, String txt) {
        SwingUtilities.invokeLater(() -> {
            try {
                SimpleAttributeSet s1 = new SimpleAttributeSet();
```

```
                StyleConstants.setBold(s1, true);
                doc.insertString(doc.getLength(), snd + ": ",
s1);
                SimpleAttributeSet s2 = new SimpleAttributeSet();
                doc.insertString(doc.getLength(), txt + "\n",
s2);
                txtArea.setCaretPosition(doc.getLength());
            } catch (BadLocationException e) {}
        });
    }

    public void addBin(String snd, String fnm, byte[] bin) {
        SwingUtilities.invokeLater(() -> {
            try {
                SimpleAttributeSet s1 = new SimpleAttributeSet();
                StyleConstants.setBold(s1, true);
                doc.insertString(doc.getLength(), snd + " [fl] "
+ fnm + "\n", s1);
                txtArea.setCaretPosition(doc.getLength());
                int rv = JOptionPane.showConfirmDialog(frm, "sv
fl " + fnm + "?", "sv", JOptionPane.YES_NO_OPTION);
                if (rv == JOptionPane.YES_OPTION) {
                    JFileChooser ch = new JFileChooser();
                    ch.setSelectedFile(new File(fnm));
                    int rr = ch.showSaveDialog(frm);
                    if (rr == JFileChooser.APPROVE_OPTION) {
                        File out = ch.getSelectedFile();
                        try (FileOutputStream fos = new
FileOutputStream(out)) {
                            fos.write(bin);
                        } catch (IOException ex) {
                            ntf("sv err: " + ex.getMessage());
                        }
                    }
                }
            } catch (BadLocationException e) {}
        });
    }

    public void updPplLst(String[] ppl) {
        SwingUtilities.invokeLater(() -> {
            pplMdl.clear();
            for (String p : ppl) {
                pplMdl.addElement(p);
            }
        });
    }
```

```
    public void ntf(String s) {
        SwingUtilities.invokeLater(() -> {
            JOptionPane.showMessageDialog(frm, s);
        });
    }
}

public class DBHNDLR {
    public Connection crtCNNTN(String hst, String prt, String db,
String usr, String pss) throws Exception {
        String rl = "jdbc:mysql://" + hst + ":" + prt + "/" + db;
        Connection  cn  =  DriverManager.getConnection(rl,  usr,
pss);
        return cn;
    }

    public void nstMSG(Connection cn, String sndr, String rcv,
String txt, long tm) throws Exception {
        String q = "INSERT INTO msgs (sndr, rcv, cntnt, tmstmp)
VALUES (?, ?, ?, ?)";
        PreparedStatement st = cn.prepareStatement(q);
        st.setString(1, sndr);
        st.setString(2, rcv);
        st.setString(3, txt);
        st.setLong(4, tm);
        st.executeUpdate();
        st.close();
    }

    public List<String> ldMSGS(Connection cn, String usr) throws
Exception {
        List<String> rs = new ArrayList<>();
        String q = "SELECT sndr, cntnt FROM msgs WHERE rcv=?
ORDER BY tmstmp DESC";
        PreparedStatement st = cn.prepareStatement(q);
        st.setString(1, usr);
        ResultSet r = st.executeQuery();
        while (r.next()) {
            String  ln  =  r.getString("sndr")  +  ":"  +
r.getString("cntnt");
            rs.add(ln);
        }
        r.close();
        st.close();
        return rs;
    }
}

public class LGNXTND {
```

```
    public boolean chckCRD(String usr, String pss) {
        if (usr == null || pss == null) return false;
        if (usr.length() < 3 || pss.length() < 3) return false;
        if (pss.contains(" ")) return false;
        if (usr.matches(".*[^a-zA-Z0-9].*")) return false;
        return true;
    }

    public String grnTKN() {
        String t = "";
        for (int i = 0; i < 64; i++) {
            int n = (int)(Math.random() * 62);
            if (n < 10) t += (char)('0' + n);
            else if (n < 36) t += (char)('A' + (n - 10));
            else t += (char)('a' + (n - 36));
        }
        return t;
    }

    public boolean vrfTKN(String tkn, Set<String> st) {
        if (tkn == null) return false;
        if (st.contains(tkn)) return true;
        return false;
    }
}

public class FLHNDLR {
    public void svMSG(String flpth, String sndr, String rcv,
String msg) throws Exception {
        File f = new File(flpth);
        FileOutputStream fs = new FileOutputStream(f, true);
        String ln = sndr + "->" + rcv + ":" + msg +
System.lineSeparator();
        fs.write(ln.getBytes("UTF-8"));
        fs.flush();
        fs.close();
    }

    public List<String> rdMSG(String flpth) throws Exception {
        List<String> rs = new ArrayList<>();
        File f = new File(flpth);
        FileInputStream fs = new FileInputStream(f);
        BufferedReader br = new BufferedReader(new
InputStreamReader(fs, "UTF-8"));
        String l = "";
        while ((l = br.readLine()) != null) {
            rs.add(l);
        }
        br.close();
```

```
        fs.close();
        return rs;
    }
}

public class CRPT {
    public byte[] ncr(String txt, String ky) throws Exception {
        SecretKeySpec sk = new SecretKeySpec(ky.getBytes("UTF-
8"), "AES");
        Cipher c = Cipher.getInstance("AES");
        c.init(Cipher.ENCRYPT_MODE, sk);
        return c.doFinal(txt.getBytes("UTF-8"));
    }

    public String dcr(byte[] dt, String ky) throws Exception {
        SecretKeySpec sk = new SecretKeySpec(ky.getBytes("UTF-
8"), "AES");
        Cipher c = Cipher.getInstance("AES");
        c.init(Cipher.DECRYPT_MODE, sk);
        byte[] rs = c.doFinal(dt);
        return new String(rs, "UTF-8");
    }
}

public class CNTCTSRVC {
    public void ddCNTCT(Map<String, Set<String>> mp, String u,
String c) {
        if (!mp.containsKey(u)) mp.put(u, new HashSet<>());
        mp.get(u).add(c);
    }

    public void rmCNTCT(Map<String, Set<String>> mp, String u,
String c) {
        if (!mp.containsKey(u)) return;
        mp.get(u).remove(c);
    }

    public Set<String> gtCNTCTS(Map<String, Set<String>> mp,
String u) {
        if (!mp.containsKey(u)) return new HashSet<>();
        return new HashSet<>(mp.get(u));
    }
}

public class LGGR {
    public void wrtLG(String fl, String ln) {
        try {
            FileOutputStream fs = new FileOutputStream(fl, true);
```

```
            String x = System.currentTimeMillis() + ":" + ln +
System.lineSeparator();
            fs.write(x.getBytes("UTF-8"));
            fs.flush();
            fs.close();
        } catch (Exception e) {}
    }

    public List<String> rdLG(String fl) {
        List<String> rs = new ArrayList<>();
        try {
            File f = new File(fl);
            FileInputStream fs = new FileInputStream(f);
            BufferedReader    br    =    new    BufferedReader(new
InputStreamReader(fs, "UTF-8"));
            String l = "";
            while ((l = br.readLine()) != null) {
                rs.add(l);
            }
            br.close();
            fs.close();
        } catch (Exception e) {}
        return rs;
    }
}

public class SRVRTHRD {
    public void sndBRDCST(List<Socket> sks, String msg) {
        for (Socket s : sks) {
            try {
                OutputStream os = s.getOutputStream();
                os.write((msg                                          +
System.lineSeparator()).getBytes("UTF-8"));
                os.flush();
            } catch (Exception e) {}
        }
    }

    public void sndTRGT(Socket s, String msg) {
        try {
            OutputStream os = s.getOutputStream();
            os.write((msg                                              +
System.lineSeparator()).getBytes("UTF-8"));
            os.flush();
        } catch (Exception e) {}
    }
}

public class SSSN {
```

```
    private Map<String, Long> ss = new HashMap<>();
    public void crt(String id) {
        long t = System.currentTimeMillis();
        ss.put(id, t);
    }

    public boolean chck(String id, long ttl) {
        if (!ss.containsKey(id)) return false;
        long t = ss.get(id);
        long n = System.currentTimeMillis();
        if (n - t > ttl) {
            ss.remove(id);
            return false;
        }
        ss.put(id, n);
        return true;
    }

    public void rm(String id) {
        ss.remove(id);
    }
}

public class NTFC {
    public  void  sndNTFC(String  usr,  String  msg,  Map<String,
Socket> mp) {
        if (!mp.containsKey(usr)) return;
        try {
            Socket s = mp.get(usr);
            OutputStream os = s.getOutputStream();
            os.write((msg                                          +
System.lineSeparator()).getBytes("UTF-8"));
            os.flush();
        } catch (Exception e) {}
    }
}

public class HRTBT {
    public void sndHB(Socket s) {
        try {
            OutputStream os = s.getOutputStream();
            os.write(("HB"                                         +
System.lineSeparator()).getBytes("UTF-8"));
            os.flush();
        } catch (Exception e) {}
    }

    public boolean chckHB(long lst, long lim) {
        long n = System.currentTimeMillis();
```

```
        if (n - lst > lim) return false;
        return true;
    }
}

public class FLSHRSR {
    public void snc(File src, File dst) throws Exception {
        FileInputStream fs = new FileInputStream(src);
        FileOutputStream fd = new FileOutputStream(dst);
        byte[] bf = new byte[4096];
        int r = 0;
        while ((r = fs.read(bf)) != -1) {
            fd.write(bf, 0, r);
        }
        fd.flush();
        fs.close();
        fd.close();
    }
}
```

ДОДАТОК В

Слайди презентації

Рисунок В.1 – Слайд 1

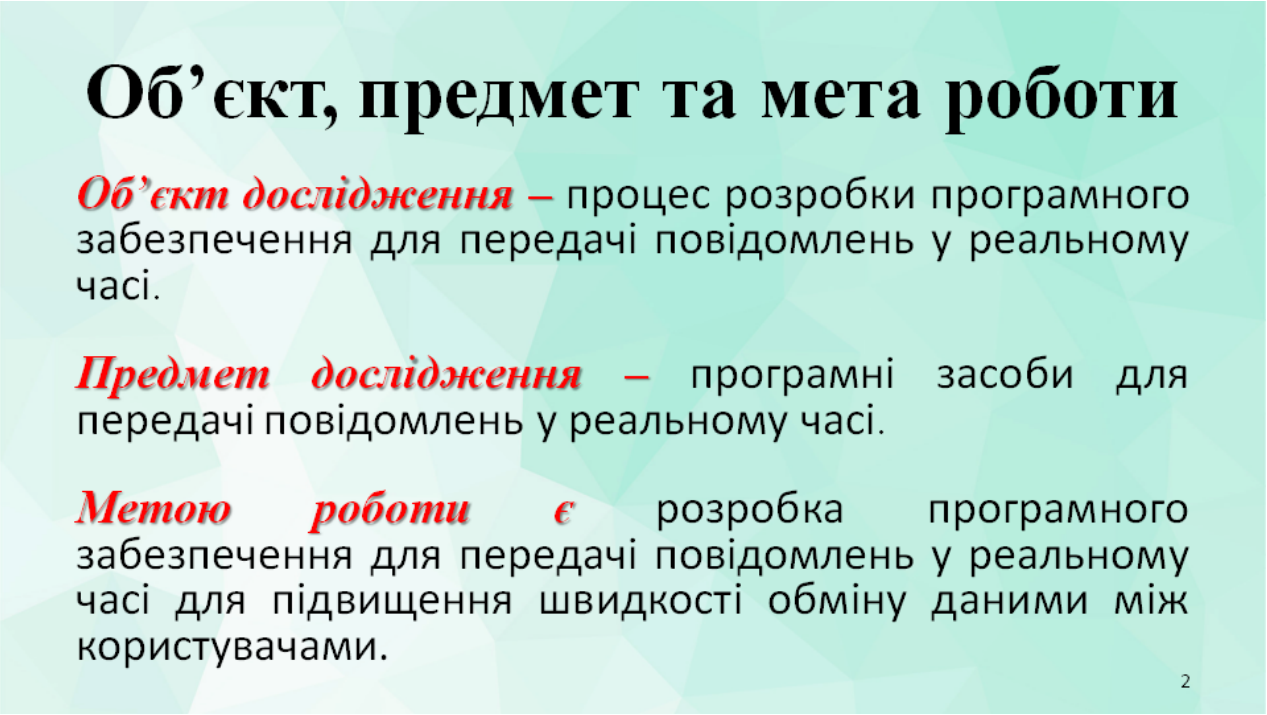

Рисунок В.2 – Слайд 2

Завдання роботи

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

– виконати аналіз предметної області та програмних засобів для передачі повідомлень у реальному часі;

– здійснити проєктування програмного забезпечення для передачі повідомлень у реальному часі;

– створити програмне забезпечення для передачі повідомлень у реальному часі;

– виконати тестування розробленого програмного забезпечення для передачі повідомлень у реальному часі.

3

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів

Критерій порівняння	Connecteam	Quidget	MirrorFly
Підтримка багатьох мов	+–	+	+
Масштабованість	+	+–	+
Орієнтація на внутрішню комунікацію	+	–	+–
Орієнтація на підтримку клієнтів	+–	+	+–
Гнучкість налаштування	+–	+–	+
Підходить для включення в інші застосунки	–	+–	+

4

Рисунок В.4 – Слайд 4

Порівняння мов програмування

Критерій порівняння мов програмування	Мова програмування		
	Java	JavaScript	Python
Продуктивність під високим навантаженням	+	+–	–
Масштабованість серверних застосунків	+	+–	+–
Розвиненість інструментів для багатопотоковості	+	–	+–
Наявність промислових рішень для обробки подій у реальному часі	+	+–	+–
Рівень безпеки та вбудованих механізмів захисту	+	+–	+–

5

Рисунок В.5 – Слайд 5

Обґрунтування вибору середовища розробки

Критерій порівняння середовищ розробки	Середовища розробки		
	Eclipse	IntelliJ IDEA	NetBeans
Зручність роботи з багатопотоковими застосунками	+	+–	+–
Підтримка інструментів аналізу продуктивності	+	+–	+
Гнучкість конфігурації та розширюваність	+	+	+–
Інтеграція з серверними платформами	+	+	+–
Потужність інструментів відлагодження	+	+	+–

6

Рисунок В.6 – Слайд 6

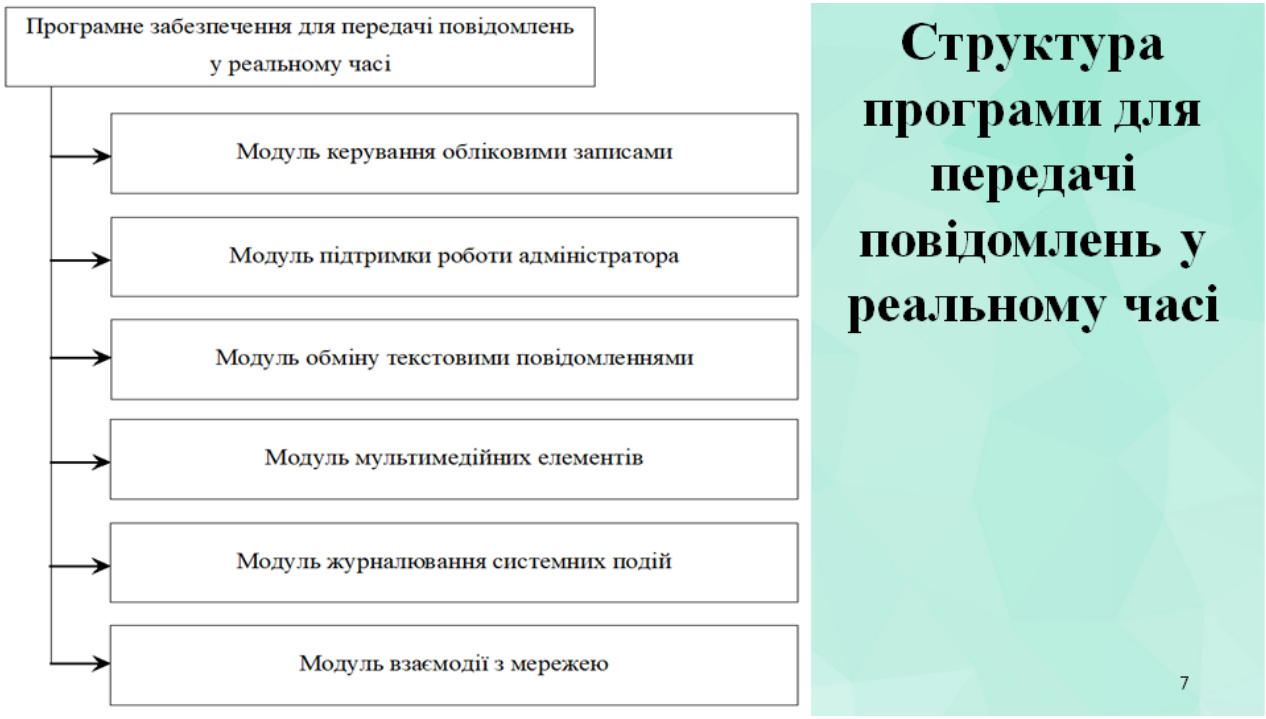

Рисунок В.7 – Слайд 7

Рисунок В.8 – Слайд 8

Рисунок В.9 – Слайд 9

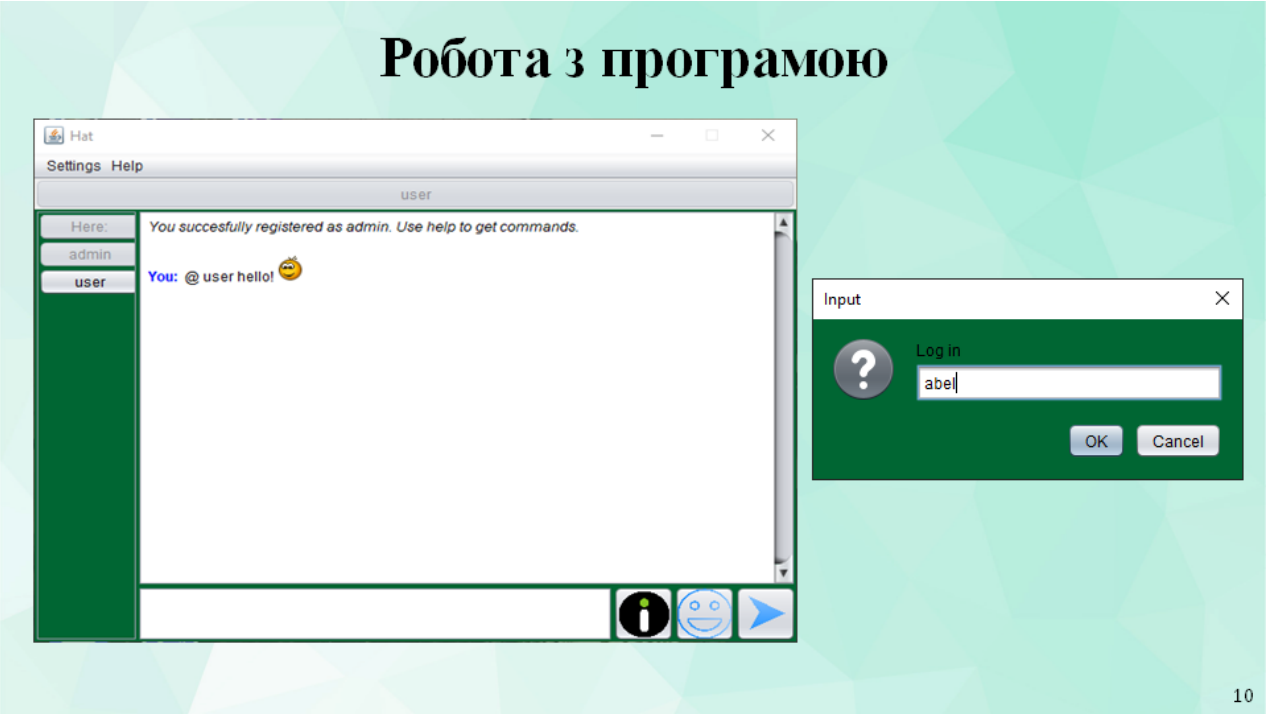

Рисунок В.10 – Слайд 10

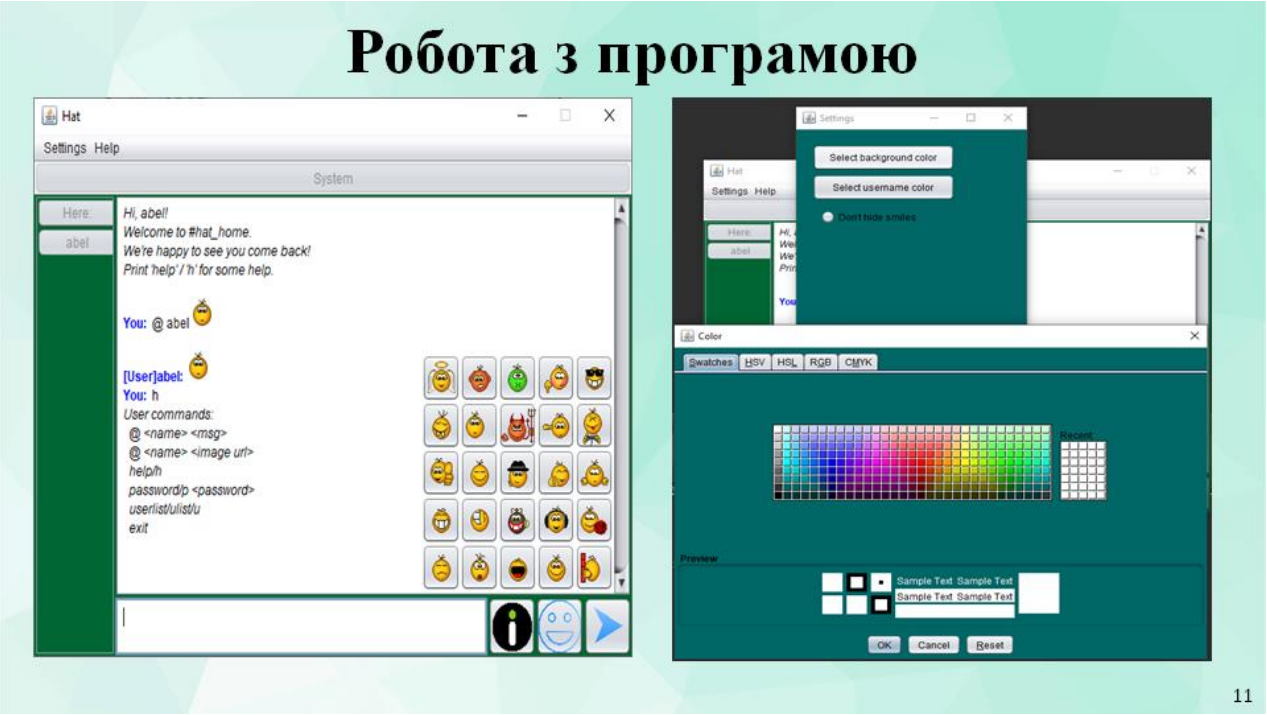

Рисунок В.11 – Слайд 11

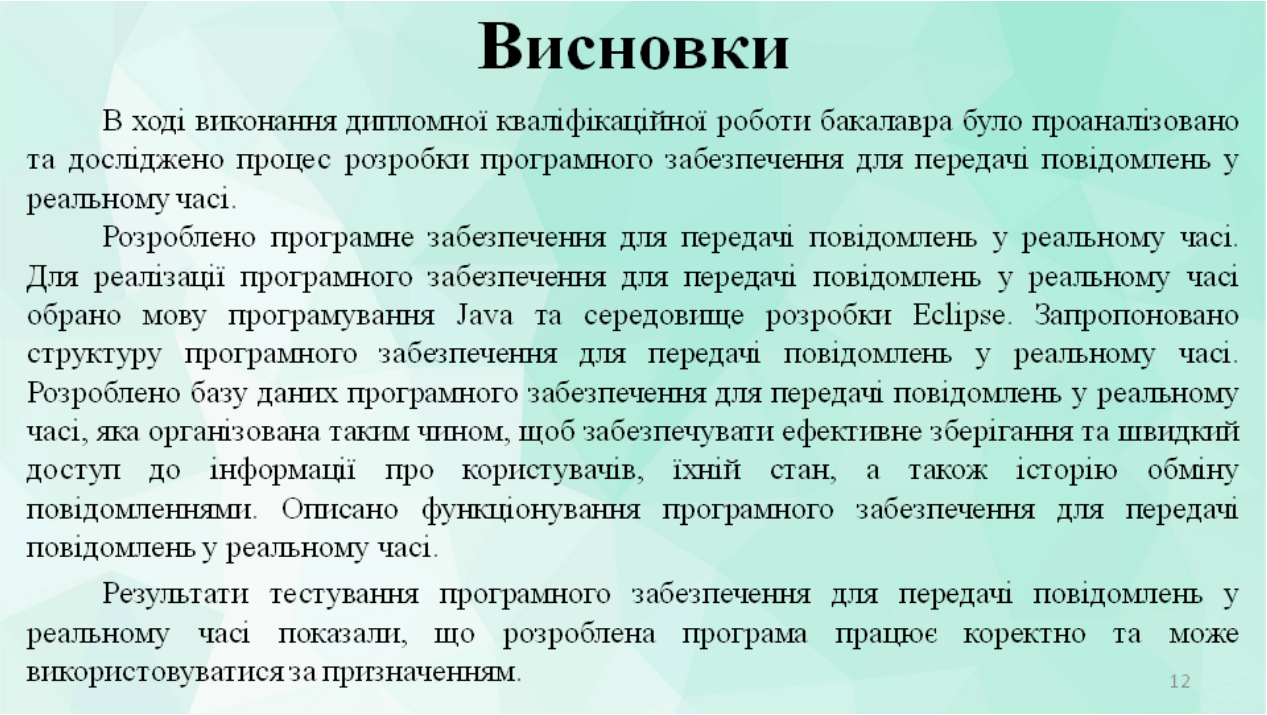

Рисунок В.12 – Слайд 12