

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ЗАКЛАД
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА”

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Нєлєпа Руслан Олександрович

**Розробка клієнт-серверного Java-додатку для автоматизації спілкування
із клієнтами**

кваліфікаційна робота

здобувача вищої освіти першого (бакалаврського) рівня
освітньої програми «Інженерія програмного забезпечення»
за спеціальністю 121 Інженерія програмного забезпечення

Особистий підпис – _____

Науковий керівник – _____
(підпис)

доцент кафедри ІТС, кандидат
педагогічних наук, доцент
С.О. Переяславська
(посада, науковий ступінь, наукове звання,
ініціали, прізвище)

Зав. кафедри – _____
(підпис)

зав. кафедри ІТС, кандидат
педагогічних наук, доцент,
М.А. Семенов
(посада, науковий ступінь, наукове звання,
ініціали, прізвище)

Полтава – 2024

Міністерство освіти і науки України

Державний заклад

„Луганський національний університет імені Тараса Шевченка”

Факультет (інститут)

Навчально-науковий інститут математики та інформаційних технологій

Кафедра, циклова комісія

Інформаційних технологій та систем

Освітній ступень

Бакалавр

Напрямок підготовки (спеціальність)

121 Інженерія програмного забезпечення
(код, назва)

Галузь знань

12, Інформаційні технології
(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

(підпис)

М.А. Семенов
(ініціали, прізвище)

“ _ ” _____ 2024 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ

Нелєпі Руслану Олександровичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка клієнт-серверного Java-додатку для автоматизації спілкування із клієнтами

Керівник кваліфікаційної роботи

Переяславська С.А., к.пед.н., доцент

(прізвище, ініціали, науковий ступінь, вчене звання)

затверджена наказом по університету

від

2. Строк подання студентом проекту (роботи)

3. Вихідні дані до роботи Клієнт-серверний додаток “Корпоративний чат”, створений на платформі Java.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) поняття, види та складові клієнт-серверних архітектур і протоколів, розробка програмного додатка на платформі програмування Java, структура та логіка клієнт-серверного додатку, етапи програмної реалізації.

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

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання „_____” _____ 2023р.

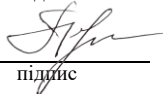
КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1.	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 24 жовтня	
2.	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	До 1 лютого	
3.	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 15 лютого	
4.	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 1 квітня	
5.	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	Перший тиждень квітня	
6.	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 31 квітня	
7.	Попередній захист роботи на кафедрі	травень	
8.	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації	
9.	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	

Студент

Керівник проекту (роботи)

підпис



підпис

Р.О. Нелепа

(ініціали, прізвище)

С.О. Переяславська

(ініціали, прізвище)

АНОТАЦІЯ

Нєлєпа Р.О.

Тема: Розробка клієнт-серверного Java-додатку для автоматизації спілкування із клієнтами

Спеціальність: 121 «Інженерія програмного забезпечення».

Установа: ДЗ Луганський національний університет імені Тараса Шевченка, 2024 р.

Кваліфікаційна робота: 50 с., 1 табл., 9 рис., 24 джерела.

Мета роботи – розробка клієнт-серверного Java-додатку “Корпоративний чат” .

Об’єкт дослідження – клієнт-серверний додаток “Корпоративний чат”

Предмет дослідження – розробка клієнт-серверних додатків на платформі програмування Java із застосування технології сокетів.

Методи дослідження: *теоретичні:* аналіз наукової літератури, узагальнення та систематизація теоретичних положень про створення клієнт-серверних додатків, додатків на мові програмування Java; *емпіричні:* порівняльний аналіз можливостей засобів розробки клієнт-серверних додатків; *експериментальні:* тестування розробленого додатка.

Результати роботи: розроблен клієнт-серверний додаток “Корпоративний чат ” на платформі Java, який може застосовуватися у навчальних цілях під час підготовки відповідних напрямків спеціалістів.

Висновки : узагальнено теоретичні підходи для розробки клієнт-серверних додатків, розглянуто види та складові клієнт-серверних архітектур. Визначено переваги платформи програмування Java. Проаналізовано етапи створення клієнт-серверного додатку "Корпоративний чат" та реалізовано на платформі Java.

Ключові слова: КЛІЄНТ-СЕРВЕРНИЙ ДОДАТОК, МОВА ПРОГРАМУВАННЯ JAVA, MVC, ЧАТ, СОКЕТ , ВНУТРІШНІ І ВКЛАДЕНІ КЛАСИ,

ABSTRACT

Nelepa R.O.

Topic: Development of a client-server Java application for automating communication with clients.

Specialty: 121 "Software Engineering".

Institution: DZ Luhansk Taras Shevchenko National University, 2024

Qualification work: 50 pages, 1 table, 9 figures, 24 sources.

The purpose of the work: development of the client-server Java application "Corporate Chat".

The object: client-server application "Corporate chat"

The subject: development of client-server applications on the Java programming platform.

Research methods: *theoretical:* analysis of scientific literature, generalization and systematization of theoretical provisions on the creation of client-server applications, applications in the Java programming language; *empirical:* comparative analysis of the possibilities of developing client-server applications; *experimental:* testing of the developed application.

Results: developed a client-server application "Corporate Chat" on the Java platform, which can be used for educational purposes in the preparation of relevant areas of specialists.

Conclusions: the theoretical approaches for development of client-server applications are generalized, types and components of client-server architectures are considered. The advantages of the Java programming platform are determined. The stages of creating a client-server application "Corporate Chat" are analyzed and implemented on the Java platform.

Keywords: CLIENT SERVER APPLICATION, JAVA PROGRAMMING LANGUAGE, MVC, CHAT, SOCKET, INNER AND NASTED CLASSES

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

М.А. Семенов

(підпис)

(ініціали, прізвище)

“ ” 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання програмної розробки (ПР):

Розробка клієнт-серверного Java-додатку для
автоматизації спілкування із клієнтами

ПОГОДЖЕНО
Керівник кваліфікаційної роботи

С.О. Переяславська

“ ” 2024р

ВИКОНАВЕЦЬ
Студент групи 4ПЗ

Нєлєпа Р.О.

“ ” 2024р

Полтава – 2024

ЗМІСТ

ВСТУП	3
1. Підстави для розробки програми.....	3
2. Призначення розробки.....	3
3. Вимоги до програми	3
3.8. Спеціальні вимоги:	5
4. Вимоги до програмної документації.....	5
5. Етапи виконання розробки.....	5
6. Порядок контролю і прийому	6
7. Порядок внесення змін до технічного завдання, що затверджено.	6

ВСТУП

Найменування програми: клієнт-серверний додаток “Корпоративний чат” на платформі програмування Java.

Область застосування: Програма призначена для використання на персональних комп'ютерах.

1. Підстави для розробки програми

1.1. Перелік документів, на підставі яких ведеться розробка:

- Технічне завдання
- Пояснювальна записка
- Програма

1.2. Організація: ЛНУ імені Тараса Шевченка.

1.3. Терміни розробки:

Початок 30 жовтня 2023 р.

Закінчення 01 травня 2024р

1.4. Умовне позначення: к-с додаток ”Корпоративний чат”

2. Призначення розробки

2.1. Функціональне призначення: Здатність клієнтів підключитися до серверу та обмінюватися текстовими повідомленнями між собою, а також можливість взаємодії з ботом, який має функцію відповідати на команди запропоновані ним при підключенні до серверу.

2.2. Експлуатаційне призначення: Програма повинна експлуатуватися на персональних комп'ютерах користувачів.

3. Вимоги до програми

3.1. Вимоги до функціональних характеристик:

3.1.1. Користувач системи повинен мати змогу виконувати наступні функції:

- Можливість редагувати та надсилати запит;
- Можливість перегляду інформації;

- Можливість обміну текстовими повідомленнями з клієнтом;
- Можливість взаємодії з ботом за допомогою використання команд;
- Можливість відстежувати процес за допомогою графічного інтерфейсу;
- Можливість взаємодіяти з консоллю та відстежувати процес;

3.2 Вимоги до надійності:

3.2.1. Програмний комплекс повинен надійно працювати.

3.2.2. Розробник гарантує роботу програмного додатку без збоїв та переналаштувань.

3.2.3. Передбачити контроль введеної інформації при підключенні для встановлення з'єднання між сервером і клієнтом, а також при під'єднанні бота.

3.3. Умови експлуатації:

3.3.1. Кінцевий користувач використовує персональний комп'ютер, на якому програмний додаток повинен надійно працювати.

3.4. Вимоги до складу та параметрів технічних засобів:

3.4.1. Процесор з тактовою частотою 1.2 ГГц і більше.

3.4.2. Відеокарта із 128 Мб відеопам'яті і вище.

3.4.3. 256 Мб оперативної пам'яті і вище.

3.4.4. 100 Мб вільного місця на накопичувачі.

3.4.5. Дісплей із розширенням 1366x768 і вище.

3.4.6. Клавіатура, комп'ютерна миша/тачпад.

3.4.7. Мережа Internet

3.5. Вимоги до інформаційної і програмної сумісності:

4.5.1. Операційна система Windows XP/Vista/7/8/8.1/10.

3.6. Вимоги до маркування та упаковки:

3.6.1. Програма поширюється може поширюватися за допомогою флеш-картки.

3.6.2. Вимоги до маркування не пред'являються.

3.7. Вимоги до транспортування і зберігання:

4.7.1. Поширення програми можливе через мережу інтернет, на зовнішніх накопичувачах тощо – на розсуд замовника.

3.7.2. Особливості зберігання відповідні до особливостей зберігання накопичувачів, на яких розміщена програма.

3.8. Спеціальні вимоги:

3.8.1. Мова програмування – Java

3.8.2. Рівень мови програмування – 6 або вище.

3.8.3. Програмна архітектура додатку: клієнт-сервер.

3.8.4. Мережевий протокол – TCP

3.8.5. Програмний інтерфейс для забезпечення обміну даними між процесами – сокет (клієнтський, серверний).

4.Вимоги до програмної документації

Перелік документів, що йдуть у комплекті з програмою:

4.1. Технічне завдання

4.2. Пояснювальна записка

5. Етапи виконання розробки

Етапи виконання можуть уточнюватись згідно календарного плану робіт по узгодженню між замовником та виконавцем.

Таблиця 1.

Етапи виконання робіт

№	Етапи виконання роботи	Термін виконання та обсяг робіт	звітні матеріали
1	Аналіз розробки клієнт-серверного додатку та розробка першої версії. Аналіз вимог. Розробка структури. Попереднє тестування	3 місяці	Демо-версія клієнт-серверного додотку на ЕОМ замовника, що виконує всі основні функції та звітна документація
2	Коректування структури. Розробка допоміжних функцій. Розробка остаточної версії додатку та його опрацювання. Тестування	2 місяці	Готовий клієнт-серверний додаток на ЕОМ замовника та звітна документація

№	Етапи виконання роботи	Термін виконання та обсяг робіт	звітні матеріали
3	Доопрацювання окремих модулів. Розробка звітних матеріалів згідно п.4 цього ТЗ	1 місяць	звітні матеріали згідно пункту 4

6. Порядок контролю і прийому

6.1. Представлення дипломної роботи до попереднього захисту

6.2. Представлення дипломної роботи до захисту

7. Порядок внесення змін до технічного завдання, що затверджено.

Дане технічне завдання може уточнюватися в процесі розробки ПР при узгодженні сторін з оформленням доповнень до ТЗ.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Факультет (інститут)

Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)

Кафедра

Кафедра інформаційних технологій та систем

(повна назва)

Пояснювальна записка
до кваліфікаційної роботи
за першим (бакалаврським) рівнем освіти

**Розробка клієнт-серверного Java-додатку для
автоматизації спілкування із клієнтами**

Виконав: студент 4 курсу
напряму підготовки (спеціальності)
121 «Інженерія програмного
забезпечення»

(шифр і назва напряму підготовки, спеціальності)

Нєлєпа Р.О.

(прізвище та ініціали)

Керівник

Переяславська С.О.

(прізвище та ініціали)

Рецензент

Козуб Ю.Г.

(прізвище та ініціали)

Полтава – 2024 року

ЗМІСТ

ВСТУП	4
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ КЛІЄНТ-СЕРВЕРНИХ ДОДАТКІВ.....	6
1.1. Клієнт-серверна архітектура	6
1.2. Моделі клієнт-серверних систем	7
1.3. Протоколи взаємодії клієнта і сервера	11
1.4. Особливості розробки клієнт-серверних додатків і їх реалізація на мові програмування Java	13
Висновки до розділу 1	16
РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ДОДАТКА “КОРПОРАТИВНИЙ ЧАТ” НА ПЛАТФОРМІ JAVA	17
2.1. Короткий огляд структури додатку	17
2.2. Розробка серверного додатку	18
2.3. Розробка клієнтського додатку.....	22
2.4. Розробка бота.....	24
2.5. Приклад роботи додатка.....	27
Висновки до розділу 2	30
ВИСНОВКИ.....	31
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	32
ДОДАТОК А.....	34
ДОДАТОК В.....	43

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

OOP – Object-Oriented Programming

MVC – Model View Controller

TCP – Transmission Control Protocol

IP – Internet Protocol

JVM – Java Virtual Machine

JRE – Java Runtime Environment

JDK – Java Development Kit

GUI – Graphical User Interface

ВСТУП

Актуальність роботи. Кожен день люди користуються сучасними інформаційними технологіями не замислюючись про те як вони влаштовані і як працюють. На даний момент більшість користувачів не уявляють свого існування без доступу до всесвітньої мережі і при цьому навіть не цікавляться тим на якому принципі і за допомогою яких інструментів здійснюється обмін інформацією. Основним бажанням більшості користувачів є можливість споживати інформацію, а також легко отримувати прості відповіді на конкретно поставлені запитання. Як показує практика в плані взаємодії, одним з найефективніших способів є підхід "клієнт-сервер" - це концепція під якою розуміють дві сторони комунікуючих між собою. Так само варто звернути увагу що в основі взаємодії по типу "клієнт-сервер" лежить принцип того, що роботу завжди починає тільки клієнт, а серверна частина тільки відгукується на його запит. Мова Java є основою практично для всіляких типів мережевих додатків. У світі налічуються мільйони фахівців, які розробляють додатки на Java, яка дозволяє ефективно розробляти та тестувати їх.

Метою роботи є розробка клієнт-серверного Java-дodatка "Корпоративний чат".

Об'єкт дослідження: клієнт-серверний додаток "Корпоративний чат" на Java з використанням сокетів.

Предмет дослідження: технологія створення клієнт-серверного додатка за допомогою об'єктно орієнтованої мови програмування Java.

Відповідно до предмета дослідження і мети, були виділені основні **завдання дослідження:**

- узагальнити поняття, види і складові при реалізації клієнт-серверного додатка на Java.
- розглянути підходи і принципи створення клієнт-серверного додатка.
- дослідити можливості Java для розробки клієнт-серверного додатку, зокрема технологію сокетів;

- розробити клієнт-серверний додаток "Корпоративний чат" на мові Java і протестувати його.

Для вирішення завдань дослідження використано такі **методи дослідження**: *теоретичні*: аналіз наукової літератури, узагальнення та систематизація теоретичних положень про створення клієнт-серверних додатків на мові програмування Java з використанням сокетів; *емпіричні*: порівняння моделей клієнт-серверних додатків; *експериментальні*: тестування розробленої програми.

До складу роботи входять два розділи, які висвітлюють питання розробки клієнт-серверного додатку на мові програмування Java.

Практичне значення розробки – розроблений клієнт-серверний додаток на мові програмування Java, який може застосовуватися у навчальних цілях під час підготовки відповідних напрямків спеціалістів та використовуватися у розважальних цілях.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ КЛІЄНТ-СЕРВЕРНИХ ДОДАТКІВ

1.1. Клієнт-серверна архітектура

У світі технологій, інтернет приніс докорінні зміни в процес обміну інформацією, який об'єднав всю земну кулю. Інтернет це мережа, і ця мережа має архітектуру і структуру для комунікації.

Модель клієнт-сервер - це розподілена структура додатку, яка розділяє завдання або робочі навантаження між постачальниками ресурсу або служби, званими серверами, і ініціаторами запитів служб, званими клієнтами. Часто клієнти і сервери обмінюються даними по комп'ютерній мережі на окремому обладнанні, і клієнт, і сервер можуть перебувати в одній системі. Хост сервера запускає одну або кілька серверних програм, які діляться своїми ресурсами з клієнтами. Клієнт зазвичай не ділиться своїми ресурсами, але запитує контент або послугу з сервера. Таким чином, клієнти ініціюють сеанси зв'язку з серверами, які очікували вхідних запитів. Прикладами комп'ютерних додатків, що використовують модель клієнт-сервер, є електронна пошта, інтернет. Клієнт-серверні обчислення - це найбільш ефективне джерело інструментів, які наділяють співробітників повноваженнями і відповідальністю.

Клієнт-серверні обчислення - це нездоланий рух, який змінює спосіб використання комп'ютерів. Хоча ці обчислення дуже молоді, вони вже використовуються в повну силу і не залишають недоторканими жодну область і куточок комп'ютерної індустрії. Розробка клієнт-серверних додатків вимагає гібридних навичок, які включають обробку транзакцій, проектування бази даних, дизайн графічного призначеного для користувача інтерфейсу і вміння працювати з Інтернетом.

Компоненти клієнт - серверної архітектури.

Архітектура клієнт-сервер працює, коли клієнтський комп'ютер відправляє запит ресурсу або процесу на сервер з мережевого з'єднання, який потім обробляється і доставляється клієнту. Серверний комп'ютер може керувати декількома клієнтами одночасно, в той час як один клієнт може бути підключений до декількох серверів одночасно, кожен з яких надає свій набір послуг. Дана архітектура складається з трьох компонентів, а саме клієнт, сервер і мережу.

Клієнт - це розрахована на одного користувача робоча станція, яка надає послуги презентацій і відповідні обчислення, можливості підключення, послуги бази даних і інтерфейси, що відповідають потребам бізнесу.

Мережа - з'єднання яке забезпечує обмін інформацією між клієнтом і сервером.

Сервер - це один або кілька багатокористувацьких процесорів із загальною пам'яттю, що забезпечують обчислення, зв'язок і служби баз даних, а також інтерфейси, що відповідають потребам бізнесу.

1.2. Моделі клієнт-серверних систем

Розрізняють три рівні архітектур "клієнт - сервер":

Архітектура Peer-to-peer - однорангова, децентралізована або пірінгова мережа. Заснована на рівноправ'ї учасників, тобто в такій мережі відсутні виділені сервери і кожен вузол є клієнтом і одночасно виконує функції сервера. У мережі присутня деяка кількість комп'ютерів, де кожен може зв'язатися з будь-яким іншим. У подібній архітектурі кожен комп'ютер повинен мати здатність обробляти запити від інших комп'ютерів в мережі.

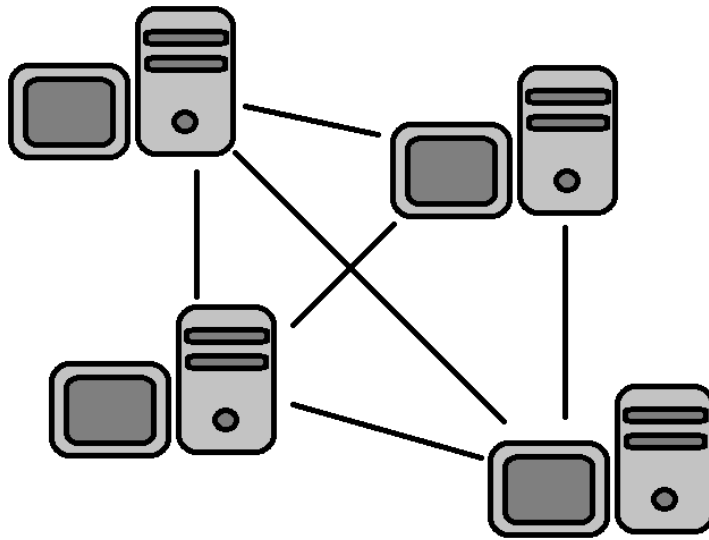


Рис.1.1. Однорагова або пірінгова архітектура

Дворівнева архітектура – термін «дворівневий» описує спосіб поділу обробки в додатку клієнт / сервер. Дворівневий додаток ідеально забезпечує кілька робочих станцій з єдиним рівнем уявлення, який взаємодіє з рівнем централізованого зберігання даних. Рівень представлення зазвичай є клієнтом, а рівень зберігання даних - сервер. Фактично, більшість архітектур клієнт / сервер є дворівневими. У подібній архітектурі, в якій мережеве навантаження розподілено між сервером і клієнтами - сервер очікує від клієнтських програм запити і надає їм свої ресурси у вигляді даних (наприклад, завантаження файлів або робота з базами даних) або ресурси у вигляді сервісних функцій (наприклад, робота з електронною поштою, спілкування за допомогою миттєвого обміну повідомленнями і даними або перегляд веб сторінок в інтернеті).

Ви можете уникнути проблем з дворівневим клієнт-сервером, розширивши два рівня до трьох. Триврівнева архітектура додає до дворівневої моделі новий рівень, який ізолює обробку даних в центральному місці і максимізує повторне використання об'єктів.

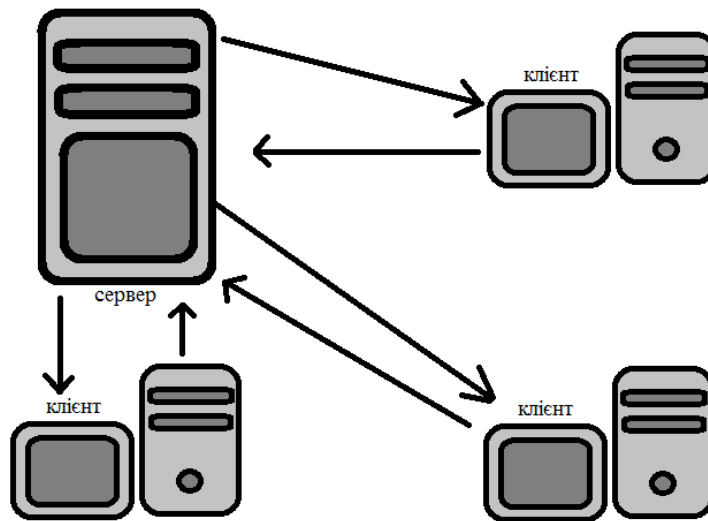


Рис. 1.2. Двурівнева архітектура

Трирівнева або багаторівнева архітектура - дана модель клієнт-серверної системи, містить у своїй архітектурі "третього учасника" - сховище даних. При використанні цього шаблону, три рівня прийнято називати шарами:

1. Клієнтський шар - програмне забезпечення з призначенням для користувача інтерфейсом, який відправляє запит на сервер і отримує відповідь.

2. Шар так званої логіки - сервер, на якому відбувається обробка запитів і відповідей. Так само його ще називають серверним шаром. Тут відбуваються всі логічні операції: операції з даними, звернення до інших сервісів або сховищ даних.

3. Шар даних - сервер баз даних, до нього звертається сервер. У цьому шарі зберігається вся необхідна інформація, якою користується додаток при роботі. Таким чином, сервер приймає на себе обов'язок по зверненню до даних, не даючи можливості користувачеві звернутися до них безпосередньо. Трирівнева архітектура використовується для великих і складних проектів з недоступними для користувача базами даних, для збереження і захисту даних.

Використовуючи таку архітектуру, отримуємо чимало плюсів, основні з них:

- можливість побудувати захист, при атаці зловмисника на сховище даних;
- можливість регулювати доступ до призначених для користувача даних;
- можливість редагувати і модифікувати дані перед відправкою їх клієнту, так само завдяки подібній системі ми отримуємо можливість скоротити обсяг інформації, який відправимо клієнту і тим самим знизимо вимоги до якості з'єднання;
- можливість розширити програму на кілька серверів, при тому що вони все ще залишатимуться з можливістю використовувати ту ж саму базу даних.

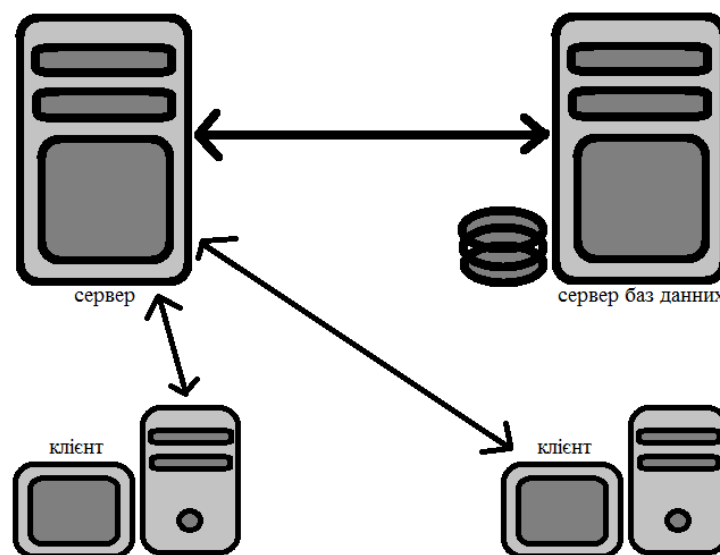


Рис. 1.3. Трирівнева архітектура

Описані вище моделі відображають сучані підходи до архітектурної побудови клієнт-серверних додатків.

1.3. Протоколи взаємодії клієнта і сервера

Протокол визначає загальний набір правил і сигналізує, що комп'ютери в мережі використовують для зв'язку. Існує безліч протоколів, кожен з яких має переваги перед іншими.

Найпопулярніші з них:

TCP / IP

Протокол управління передачею (TCP), Інтернет-протокол (IP). Спочатку він був розроблений Міністерством оборони США, щоб дозволити різним комп'ютерам спілкуватися. Сьогодні, як багато хто з нас знають, цей протокол використовується в якості основи для Інтернету. Оскільки TCP / IP повинен охоплювати такі великі відстані і перетинати кілька невеликих мереж, він є маршрутизації протоколом, що означає, що він може відправляти дані через маршрутизатор по дорозі до місця призначення. В кінцевому підсумку це трохи уповільнює роботу, але ця можливість робить її дуже гнучкою для великих мереж.

IPX / SPX

Розроблено компанією Novell. IPX схожий на оптимізований TCP / IP.

Це теж маршрутизації протокол, що робить його зручним для великих мереж, але він забезпечує більш швидкий доступ по мережі, ніж TCP / IP. Недоліком є те, що він погано працює по аналогових телефонних лініях. IPX постійно перевіряє статус передачі, щоб бути впевненим, що всі дані надходять. Це вимагає додаткової смуги пропускання, коли у аналогових телефонних ліній для початку не так багато. Це призводить до повільного доступу. Звичайно, з IPX дані більш надійні.

NetBEUI

Цей протокол, розроблений Microsoft, розроблений для невеликих локальних мереж і працює досить швидко. У ньому відсутні накладні витрати на адресацію TCP / IP і IPX, що означає, що його можна використовувати

тільки в локальних мережах. Ви не можете отримати доступ до мереж через маршрутизатор.

Архітектура TCP / IP

Набір протоколів TCP / IP складається з наступних компонентів: мережевий протокол (IP) і його логіка маршрутизації, три транспортних протоколу (TCP, UDP і ICMP), а також ряд служб сеансів, презентацій і додатків.

IP представляє мережевий рівень. Кожній системі призначається унікальний мережевий адрес, незалежно від того, чи підключена вона до локальної або глобальної мережі. TCP надає транспортні послуги по IP. Він орієнтований на з'єднання, тобто вимагає, щоб між двома сторонами був встановлений сеанс для надання своїх послуг. Він забезпечує наскрізну передачу даних, усунення помилок, упорядкування даних і управління потоком. TCP забезпечує той вид зв'язку, який користувачі та програми очікують в локально підключених сеансах. Цікаво, що навколишнє середовище взаємозалежної локальної мережі демонструє багато з тих же характеристик, що і середовище, для якої був розроблений TCP / IP.

Зокрема:

Маршрутизація: міжмережам потрібна підтримка маршрутизації; маршрутизація дуже ефективна в середовищах TCP / IP.

З'єднання в порівнянні з режимом без встановлення з'єднання: активність LAN включає обидва; то набір протоколів TCP / IP ефективно підтримує обидва в рамках інтегрованої структури.

Чутливість до адміністративного навантаження: адміністративна підтримка LAN зазвичай обмежена. Середовища TCP / IP містять величезну кількість динамічних можливостей, в яких пристрої та мережі виявляються динамічно, а таблиці маршрутизації автоматично підтримуються і синхронізуються.

Мережі мереж: TCP / IP забезпечує виняткову гнучкість в якості адміністративного підходу до управління об'єднаннями мереж.

Використовуючи переваги свого динамічного характеру, він забезпечує дуже незалежне управління частинами мережі (при необхідності).

1.4. Особливості розробки клієнт-серверних додатків і їх реалізація на мові програмування Java

Java - це мова програмування, винайденна Sun Microsystems. Мета Sun полягала в тому, щоб дозволити програмістам створити одну копію програми, яку користувачі могли б запускати практично на будь-якому комп'ютері і операційній системі. Ця можливість була розроблена, щоб зробити Java життєво важливим компонентом програмування в Інтернеті.

Програми, написані на Java, переводяться в свою власну архітектуру в форматі, званому байт-кодами. Щоб запустити програму Java, користувачеві потрібна інша програма, яка може інтерпретувати програму Java і надати їй необхідне середовище і послуги. Програмний рівень, віртуальна машина Java (JVM), робить практично будь-яку апаратну і програмну платформу схожою на програму Java. Фактично, JVM - це драйвер пристрою для програм Java. Запуск Java-програми з JVM в даний час повільніше, ніж запуск еквівалентної C-програми, але технологія JVM поліпшується швидкими темпами.

Архітектура клієнт-сервер має різні характеристики, які спрощують комунікацію. Модель клієнт-сервер є однією з центральних ідей мережових обчислень. Велика кількість додатків, використовують модель клієнт-сервер, включаючи основну програму інтернету, TCP / IP.

Деякі особливості архітектури клієнт-сервер:

- обмін повідомленнями: клієнт - сервер це система, яка слабо пов'язана і взаємодіє через механізм передачі повідомлень. Повідомлення працює як механізм доставки запитів і відповідей;

- масштабованість: можливість модифікувати дані перед відправкою їх клієнту;
- аутентифікація;
- надійність: сервер не повинен очікувати, що клієнтський код буде вільним від помилок;
- загальні ресурси: сервер має можливість обслуговувати клієнтів одночасно і регулювати їх загальний доступ до загальних ресурсів.

Система клієнт-сервер стала архітектурою обчислень і додатків по всьому світу. Тобто система клієнт - сервер наближає обробку додатків, саме, до користувача, а так само допомагає поліпшити продуктивність.

Зв'язок між клієнтом і сервером з використанням сокетів.

Серед понять і терменів, пов'язаних з роботою в мережі, в Java є таке важливе поняття як «сокет». Воно позначає точку, через яку відбувається з'єднання. Використання сокетів дозволяє створити підключення двох комп'ютерів. Простіше кажучи, сокет з'єднує в мережі дві програми.

Також, сокети дозволяють програмісту розглядати мережеве з'єднання як ще один потік, в який можна записувати або читати байти. Історично сокети є розширенням однієї з ідей UNIX: усе введення-виведення має виглядати для програмістів як файловий ввід-вивід, незалежно від того, чи працюють вони з клавіатурою, графічним дисплеєм, звичайним файлом або підключення до мережі. Сокети захищають програміста від низькорівневих деталей мережі, таких як типи носіїв, розміри пакетів, повторна передача пакетів, мережеві адреси і багато іншого. Сокет може виконувати наступні основні операції:

- підключити до віддаленій машині;
- забезпечувати обмін даними;
- закривати з'єднання;
- зв'язуватися з портом;
- очікувати сигнал входу даних;

- приймати з'єднання з віддалених машин на прив'язаному порті.

Припускаючи, що ми використовуємо TCP для реалізації, орієнтованої на з'єднання, Java має два пов'язаних класу, які необхідні, тобто клас Socket і клас ServerSocket. Клас сокета використовується як клієнтом, так і сервером і має методи, відповідні першим чотирьом з цих операцій. Останні три необхідні тільки сервера і реалізуються класом ServerSocket. Клас Socket реалізує ідею сокета. Через його канали, введення і виведення, будуть спілкуватися клієнт з сервером, по суті це так звані "розетки". Оголошується цей клас на стороні клієнта, а сервер відтворює його, отримуючи сигнал на підключення. Так відбувається спілкування в мережі. Тому, найпростішим варіантом реалізації клієнт - серверного додатка на Java буде на основі сокетів.

У цьому методі і сервер, і клієнт є додатком Java. Клієнтський додаток повинен бути завантажено на клієнтський комп'ютер. Сервер може працювати одночасно, тобто обслуговувати більше одного клієнта одночасно. Сервер і база даних знаходяться на сервері. Основним недоліком цього методу, як уже говорилося раніше, є те, що програмне забезпечення для клієнта має бути фізично присутнім на клієнтській машині, а клієнтська машина повинна мати інтерпретатор Java для його запуску. Якщо в клієнтську програму вносяться певні зміни, його необхідно поширити серед клієнтів.

Висновки до розділу 1

У данному розділі була розглянуті теоретичні основи розробки клієнт-серверних додатків.

Під час дослідження було розглянуто клієнт-серверну архітектуру, визначено ролі компонентів (клієнта, сервера) в ній, розглянуто основні моделі (однорангова, дворангова, трьохрангова).

Також проаналізовано основні протоколи, які можуть застосовуватися у клієнт-серверних архітектурах (TCP / IP, IPX / SPX, NetBEUI).

Крім того, розглянуто основні можливості мови програмування Java в розробці клієнт – серверного додатку та визначено способи реалізації на мові програмування Java майбутнього додатку.

Таким чином, визначено, що додаток «Корпоративний чат» буде мати клієнт-серверну архітектуру, із застосуванням мережевого протоколу TCP та технологією сокетів як інтерфейсу взаємодії клієнта та сервера для можливості обміну текстовими повідомленнями.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ДОДАТКА “КОРПОРАТИВНИЙ ЧАТ” НА ПЛАТФОРМІ JAVA

2.1. Короткий огляд структури додатку

В ході розробки був структурований існуючий матеріал і на його основі реалізовано клієнт-серверний додаток. На початку розробки необхідно вибрати між дворівневою і багаторівневою архітектурою. Для створення клієнт-серверного додатка була обрана дворівнева архітектура і об'єктно-орієнтована мова програмування - Java, для написання всієї програми. Вся обробка даних відбувається на сервері. Дворівневу архітектуру легко можна реалізувати. Структурна модель будувалася таким чином що б на будь-якому етапі була можливість змінювати і додавати новий функціонал.

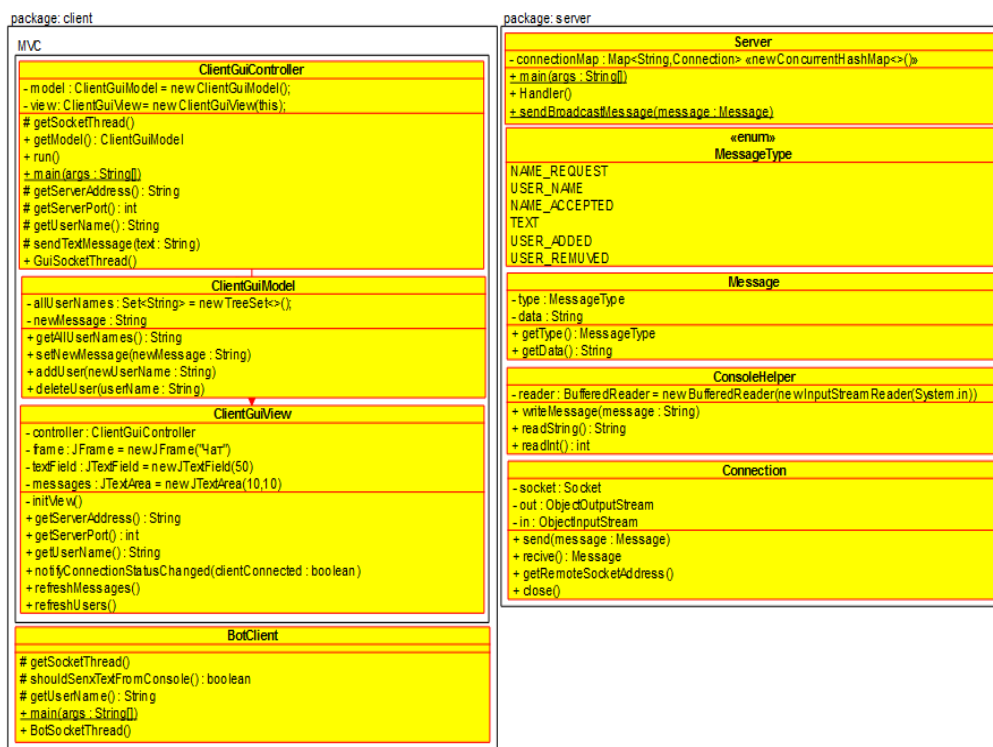


Рис.2.1. Структурна модель додатка "Корпоративний чат".

А саме ця модель складається з двох модулів client і server. До клієнтської частини відносяться такі класи як: class ClientGuiController, class ClientGuiModel, ClientGuiView, BotClient. До серверної частини відносяться

класи: `class Server`, `enum MessageType`, `class Message`, `class ConsoleHelper`, `class Connection`. Подібна структурна модель демонструє поля, члени класів і методи для роботи з ними в клієнтській і серверній частинах.

2.2. Розробка серверного додатку

Найбільш значущим у сервері є код його основного класу "Server". Розглянемо основні поля, методи класу "Server":

Сервер повинен підтримувати безліч з'єднань з різними клієнтами одночасно.

Це можна реалізувати за допомогою наступного алгоритму:

- Сервер створює серверне сокетне з'єднання.
- У циклі очікує, коли якийсь клієнт підключиться до сокету.
- Створює новий потік обробник `Handler`, в якому буде відбуватися обмін повідомленнями з клієнтом.
- Чекає наступне з'єднання.

Оскільки сервер може одночасно працювати з декількома клієнтами, нам знадобиться метод для відправки повідомлення відразу всім.

Додамо до класу `Server`:

1. Статичне поле `Map <String, Connection> connectionMap`, ключем буде ім'я клієнта, а значення - з'єднання з ним.

```
private static Map <String, Connection> connectionMap = new  
ConcurrentHashMap <> ();
```

2. Статичний метод `void sendBroadcastMessage (Message message)`, який повинен відправляти повідомлення `message` усім з'єднанням з `connectionMap`. Якщо під час надсилання повідомлення трапиться виключення `IOException`, потрібно відловити його і повідомити користувачеві що не змогли відправити повідомлення.

```
public static void sendBroadcastMessage(Message message) {  
    for (Connection connection : connectionMap.values()) {
```

```

        try{
            connection.send(message);
        }catch (IOException exception){
            ConsoleHelper.writeMessage("Не змогли отправить сообщение " +
connection.getRemoteSocketAddress());
        }
    }
}
}
}

```

Не менш важливим методом є `void notifyUsers (Connection connection, String userName) throws IOException {}`, він повинен відправляти через `connection` повідомлення про те, що був доданий користувач з ім'ям `name` для кожного імені з `connectionMap`.

```

private void notifyUsers(Connection connection, String userName) throws
IOException{
    for(String name : connectionMap.keySet()){
        if(name.equals(userName)){
            continue;
        }
        connection.send(new Message(MessageType.USER_ADDED, name));
    }
}

```

Також додамо метод який буде виконувати цикл обробки повідомлень сервером `void serverMainLoop(Connection connection, String userName) throws IOException, ClassNotFoundException`, де значення параметрів таке ж, як і у методі `notifyUsers()`.

Метод `serverMainLoop()` повинен в нескінченному циклі отримувати повідомлення від клієнта, якщо повідомлення, отримане методом `serverMainLoop()`, має тип `MessageType.TEXT`, то повинно бути відправлено нове повідомлення всім учасникам чату використовуючи метод

sendBroadcastMessage(). Якщо повідомлення отримане предметом serverMainLoop(), має тип відмінний від MessageType.TEXT, метод sendBroadcastMessage() не повинен бути викликаний, і має бути виведено повідомлення про помилку.

```
private void serverMainLoop(Connection connection, String userName)throws
IOException, ClassNotFoundException{
    while(true){
        Message message = connection.receive();
        if(message.getType() == MessageType.TEXT){
            String data = message.getData();
            sendBroadcastMessage(new Message(MessageType.TEXT, userName + ":
" + data));
        }else{
            ConsoleHelper.writeMessage("Получено сообщение от " +
socket.getRemoteSocketAddress() + ". Тип сообщения не соответствует
протоколу.");
        }
    }
}
```

Клас Handler реалізує протокол спілкування з клієнтом, за допомогою основного методу класу: run() - який буде викликати усі допоміжні методи.

1. Метод run () виводить на екран повідомлення про те що було встановлено з'єднання з віддаленою адресою.

2. створює нове з'єднання (connection) використовуючи в якості параметра поле socket.

3. Викликає метод serverHandshake () використовуючи в якості параметра щойно створене з'єднання.

4. Викликає метод sendBroadcastMessage () використовуючи в якості параметра нове повідомлення.

5. Викликає метод `notifyUsers ()` використовуючи в якості параметрів `connection` і `userName`.

6. В кінці метод `run ()` видаляє з `connectionMap` запис відповідну `userName`, і відправляє всім учасникам чату повідомлення про те, що поточний користувач був видалений.

7. Коректно обробляє виключення `IOException` і `ClassNotFoundException`.

```
public void run() {  
    ConsoleHelper.writeMessage("Установлено новое соединение с " +  
socket.getRemoteSocketAddress());  
  
    String userName = null;  
    try(Connection connection = new Connection(socket)){  
        userName = serverHandshake(connection);  
        // повідомляємо всім учасникам що приєднався новий учасник  
        sendBroadcastMessage(new Message(MessageType.USER_ADDED ,  
userName));  
        // повідомляємо новому учаснику про існуючих учасників  
        notifyUsers(connection, userName);  
        // обробляємо повідомлення користувачів  
        serverMainLoop(connection , userName);  
    } catch (IOException | ClassNotFoundException exception){  
        ConsoleHelper.writeMessage("Ошибка при обмене данными с " +  
socket.getRemoteSocketAddress());  
    }  
    if(userName != null){  
        connectionMap.remove(userName);  
        sendBroadcastMessage(new Message(MessageType.USER_REMOVED ,  
userName));  
    }  
}
```



```
}  
ConsoleHelper.writeMessage("Соединение с " +  
socket.getRemoteSocketAddress() + " закрыто.");  
}
```

2.3. Розробка клієнтського додатку

Клієнт, повинен запросити у користувача адресу і прот сервера, під'єднатися зазначеною адресою, отримати запит імені від сервера, запитати ім'я у користувача, відправити ім'я користувача сервера, дочекатися прийняття імені сервером. Після цього клієнт може обмінюватися текстовими повідомленнями з сервером. Обмін повідомленнями відбувається в двох паралельно працюючих потоках. Один займається читанням з консолі і відправкою прочитаного сервера, а другий потік отримує дані від сервера і виводить їх в консоль.

Так як клієнт реалізований з графічним інтерфейсом, то доречно було скористатися патерном MVC (Model-View-Controller). При розробці систем з призначеним для користувача інтерфейсом, слідуючи паттерну MVC потрібно розділяти систему на три складові частини. Їх, в свою чергу, можна називати модулями або компонентами. У кожній складовій компоненти буде своє призначення.

Model - так звана модель. Вона містить всю логіку програми.

View - вид, друга частина системи. Даний модуль відповідає за відображення даних користувачеві. Все, що бачить користувач, генерується видом. Controller - контролер. У ньому зберігається код, який відповідає за обробку дій користувача (будь-яка дія користувача в системі обробляється в контролері).

Model(Модель) - найбільш незалежна частина системи. Модель не повинна нічого знати про модулі Вид і Контролер. Модель настільки

незалежна, що її розробники можуть практично нічого не знати про Вид і Контролер.

Основне призначення View(Виду) - надавати інформацію з Моделі в зручному для сприйняття користувача форматі. Основне обмеження Виду - він ніяк не повинен змінювати модель. Основне призначення Контролера - обробляти дії користувача. Саме через Контролер користувач вносить зміни в модель. Точніше в дані, які зберігаються в моделі. За допомогою MVC, можна спростити написання програм, підвищити читаність коду, зробити легше розширення і підтримку системи в майбутньому. З усього цього можна зробити цілком логічний висновок. Будь-яку систему потрібно розбивати на модулі. Найбільш значущім класом клієнта є “ClientGuiController” – цей клас відповідає за обробку дій користувача (будь-яка дія користувача в системі обробляється в контролері). Розглянемо основні методи клієнтського класу “ClientGuiController”:

Вкладення клас GuiSocketThread в якому перевизначені важливі методи:

1. void processIncomingMessage (String message) - встановлює нове повідомлення моделі і викликає оновлення виведення повідомлень у представника.

```
protected void processIncomingMessage(String message) {  
    // Виводимо текст повідомлення  
    model.setNewMessage(message);  
    view.refreshMessages();  
}
```

2. void informAboutAddingNewUser (String userName) - додає нового користувача в модель і викликає оновлення виведення користувачів у відображення.

```
protected void informAboutAddingNewUser(String userName) {  
    //виводимо інформацію про додавання нового користувача  
    model.addUser(userName);  
}
```

```
view.refreshUsers();  
}
```

3. `void informAboutDeletingNewUser (String userName)` - видаляє користувача з моделі і викликає оновлення виведення користувачів у відображення.

```
protected void informAboutDeletingNewUser(String userName) {  
    //виводимо актуальний список користувачів  
    model.deleteUser(userName);  
    view.refreshUsers();  
}
```

4. `void notifyConnectionStatusChanged (Boolean clientConnected)` - викликає аналогічний метод у уявлення.

```
protected void notifyConnectionStatusChanged(boolean clientConnected) {  
    view.notifyConnectionStatusChanged(clientConnected);  
}
```

2.4. Розробка бота

Бот є клієнтом, який автоматично відповідає на деякі команди (час, дату). Тому реалізований бот відправляє поточний час і дату. Розглянемо основні методи класу `BotClient`, а саме `getUserName()` і `processIncomingMessage(String message)`:

1. `getUserName ()` - генерує нове ім'я бота на випадок, якщо до сервера підключиться кілька ботів, наприклад: `date_bot_x`, де `x` - будь-яке число від 0 до 99.

```
protected String getUserName() {  
    return "date_bot_" + (int) (Math.random() * 100); }
```

2. `processIncomingMessage (String message)` - обробляє вхідні повідомлення:

- а) Виводить в консоль текст отриманого повідомлення (message).
- б) Отримує з message ім'я відправника і текст повідомлення.
- в) Відправляє відповідь в залежності від тексту прийнятого повідомлення,

Наприклад, якщо текст повідомлення:

"Дата" - відправить повідомлення яке містить поточну дату.

"День" - відправить дату (тільки поточний день).

"Місяць" - відправить дату (тільки поточний місяць).

"Рік" - відправить дату (тільки поточний рік).

"Час" - відправить поточний час.

"Час" - відправить поточну годину.

"Хвилини" - відправить поточну хвилину.

"Секунди" - відправить поточну секунду.

Відповідь містить ім'я клієнта, який надіслав запит і чекає на відповідь.

```
protected void processIncomingMessage(String message) {  
    // Виводимо текст повідомлення в консоль  
    ConsoleHelper.writeMessage(message);  
    // Відокремлюємо відправника від тексту повідомлення  
    String userNameDelimiter = ":";  
    String[] split = message.split(userNameDelimiter);  
    if (split.length != 2) return;  
    String messageWithoutUserName = split[1];  
    // Готуємо формат для відправки дати згідно із запитом  
    String format = null;  
    switch (messageWithoutUserName) {  
        case "дата":  
            format = "d.MM.YYYY";  
            break;  
        case "день":  
            format = "d";
```

```

        break;
    case "месяц":
        format = "MMMM";
        break;
    case "год":
        format = "YYYY";
        break;
    case "время":
        format = "H:mm:ss";
        break;
    case "час":
        format = "H";
        break;
    case "минуты":
        format = "m";
        break;
    case "секунды":
        format = "s";
        break;
    }
    if (format != null) {
        String answer = new
SimpleDateFormat(format).format(Calendar.getInstance().getTime());
        BotClient.this.sendMessage("Информация для " + split[0] + ": " +
answer);
    }
}

```

2.5. Приклад роботи додатка

Java, строга мова на якій можна вирішувати будь-яке завдання, завдяки кроссплатформенності може працювати на різних платформах що дуже важливо для клієнтської частини.

Алгоритм роботи системи програми виглядає наступним чином:

1.Сервер запускається і чекає введення порту сервера;

```
Введіть порт сервера:  
8080  
Чат Сервер запущен.
```

Рис. 2.2. Запуск сервера

2.Далі після введення порту запущений сервер переходить в режим очікування підключення клієнта;

3. При підключенні клієнта, запускається призначений для користувача інтерфейс;

4.Далі сервер запитує адресу, порт і ім'я клієнта(рис.2.2., рис.2.3., рис.2.4.);

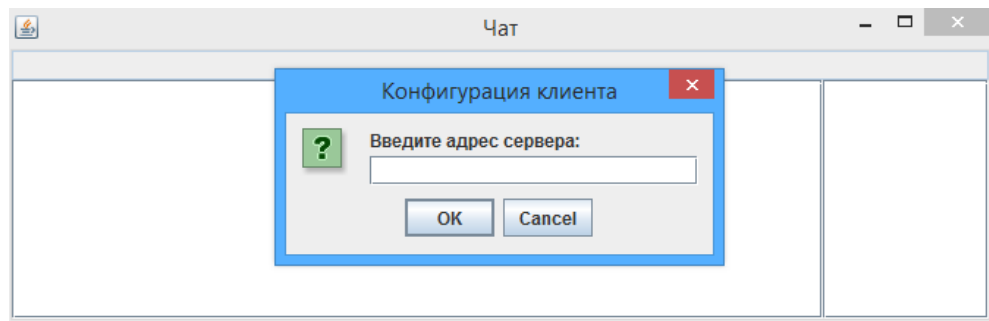


Рис.2.2. Сервер запитує адресу

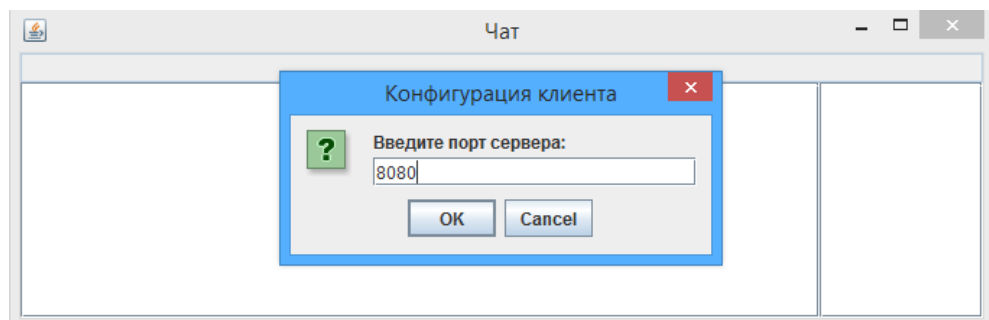


Рис.2.3. Сервер запитує порт

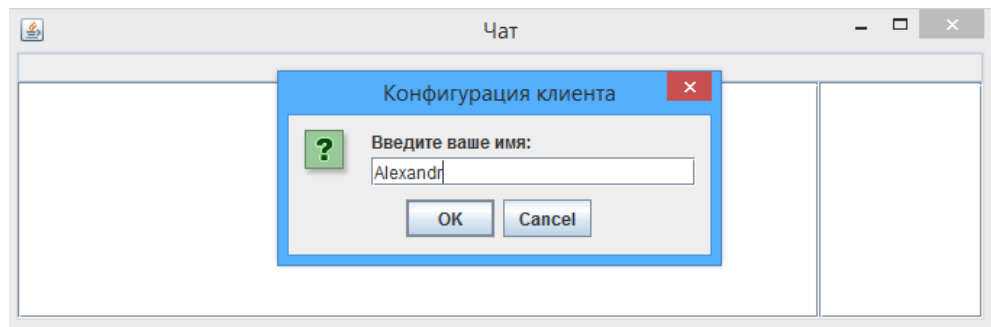


Рис.2.4. Сервер запитує ім'я

5.Після успішного з'єднання сервер обробляє отриману інформацію(рис.2.5., рис.2.6.);

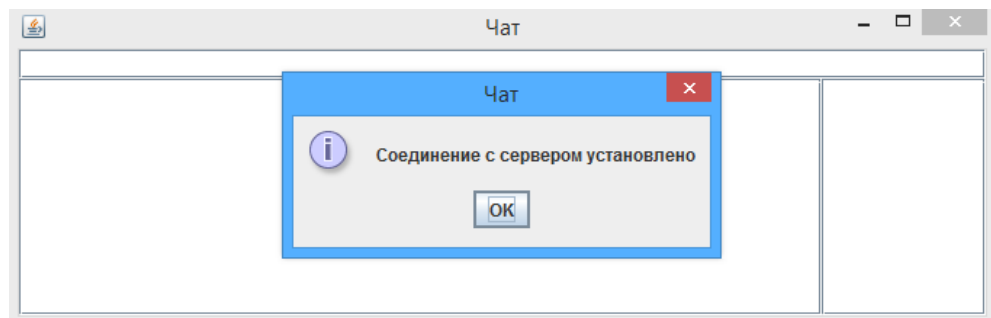


Рис.2.5. Сервер обробляє інформацію після успішного з'єднання

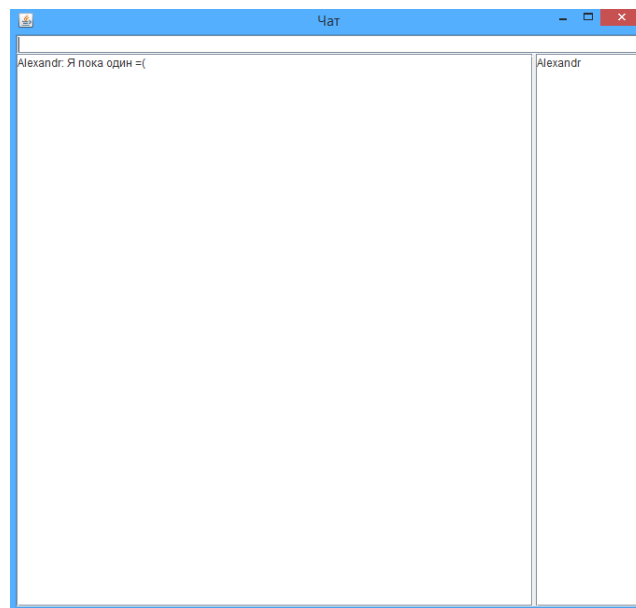


Рис.2.6. Сервер успішно з'єднався

6.Далі відбувається підключення Бот-клієнта і сервер запитує адресу і порт(рис.2.7.);

```
Введите адрес сервера:  
localhost  
Введите порт сервера:  
8080  
Участник 'date_bot_74' присоединился к чату.  
Участник 'Alexandr' присоединился к чату.  
date_bot_74: Привет чатику. Я бот. Понимаю команды: дата, день, месяц, год, время, час, минуты, секунды.
```

Рис.2.7. Підключення бота і сервер запитує адресу і порт

7. Після успішного підключення Бот-клієнт виводить інформацію про те які функції він може виконувати при введенні певних команд(рис.2.8.).

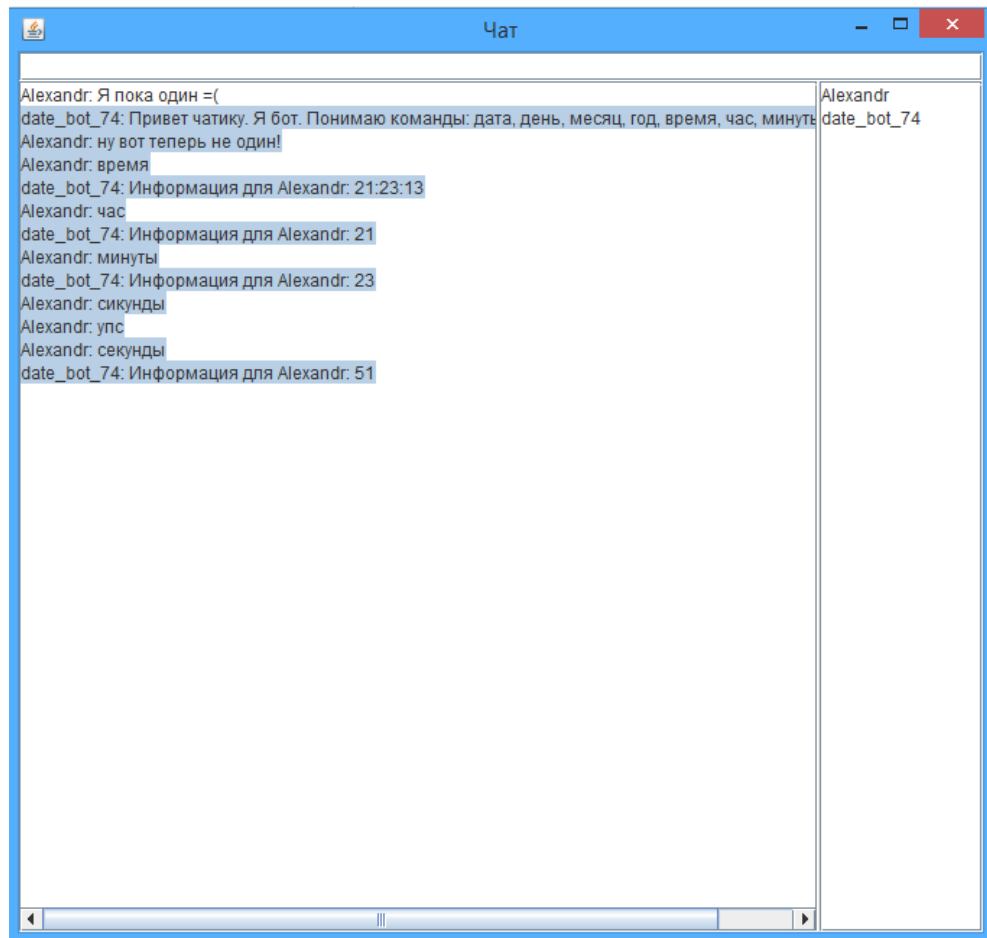


Рис.2.8. Вивід ботом інформацію про свій функціонал

Таким чином, розроблено клієнт-серверний додаток “Корпоративний чат” та протестовано його інтерфейс та функціональні можливості (додаток А).

Висновки до розділу 2

В результаті розробки була спроектована структурна модель додатка “Корпоративний чат” аналіз якого дозволив розробити, клієнт-серверний додаток за допомогою об'єктно-орієнтованої мови програмування Java.

1. Для Розробки клієнт-серверного додатку було обрано дворівневу архітектуру.
2. Розроблений клієнт-серверний додаток на мові Java, а саме:
 - для оптимізації роботи програми, був використан набір архітектурних ідей і принципів для побудови складних систем з призначенням для користувача інтерфейсом, тобто MVC;
 - розроблений сервер для обміну текстовими повідомленнями, та розглянуті основні методи класу Server;
 - розроблен клієнт з графічним інтерфейсом і можливістю підключатися до сервера і обмінюватися повідомленнями між іншими учасниками та основні методи класу ClientGuiController;
 - розроблений бот клієнт, який може приймати запити і відправляти дані про поточну дату і час.

Таким чином, був розроблен клієнт-серверний додаток “Корпоративний чат” на платформі Java.

Після розробки клієнт-серверного додатку було виконано тестування проекту, метою якого було виявлення недоліків розробки. На підставі результатів тестування можна стверджувати, що розроблений додаток є оптимізованим і працюючим.

ВИСНОВОК

Під час дослідження було проаналізовано науково-технічний матеріал, аналіз якого дозволив розробити клієнт-серверний додаток "Корпоративний чат" на платформі Java. При цьому було отримано наступні висновки та результати:

У першому розділі даної роботи був проведений огляд існуючих архітектур клієнт-серверних додатків, основних компонентів архітектур, моделі, протоколи та особливості в розробці додатків на платформі програмування Java. Проведені дослідження надали можливість визначити, що додаток «Корпоративний чат» буде мати клієнт-серверну архітектуру, із застосуванням мережевого протоколу TCP та технологією сокетів як інтерфейсу взаємодії клієнта та сервера для можливості обміну текстовими повідомленнями.

В другому розділі роботи було розроблено клієнт-серверний додаток "Корпоративний чат", а саме: логіка роботи, основні компоненти, реалізована додаткова можливість взаємодії з ботом, який надає набір команд для отримання даних про поточний час тощо. В ході реалізації програми були розглянуті і описані основні класи в розробленому додатку. Серверна частина програми створена з можливістю взаємодії з нею за допомогою консолі, клієнтська частина, реалізація якого виконана за допомогою архітектури MVC, створена з можливістю взаємодії у вигляді інтерфейсу.

В результаті, розроблений клієнт-серверний додаток може використовуватися в навчальному контексті, в наслідок чого програма має можливість доопрацьовуватися, додаючи новий функціонал з можливістю рефакторінга коду який служить метою поліпшити якість коду без зміни зовнішньої поведінки програми.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт Java™ Platform, Standard Edition 7 API Specification : веб-сайт. URL: <https://docs.oracle.com/javase/7/docs/api/overview-summary.html> (дата звернення 20.12.2020 р.)
2. Java : веб-сайт. URL: <https://uk.wikipedia.org/wiki/Java> (дата звернення 20.12.2020 р.)
3. Client-server model : веб-сайт. URL: https://en.wikipedia.org/wiki/Client-server_model. (дата звернення 15.12.2020 р.)
4. Офіційний сайт IntelliJ IDEA. URL: <https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html> (дата звернення 10.12.2020 р.)
5. GitHub : веб-сайт. URL: <https://github.com/enhorse/java-interview/blob/master/README.md#java-collections> (дата звернення 20.12.2020 р.)
6. Блох Д. Java: ефективне програмування, 3-є изд. М.: Лори, 2019. 463с.
7. Мартін Р. Чистий код: створення, аналіз і рефакторинг. К.: Фабула, 2019. 416 с.
8. Герберт Шілдрт, Java 8. Повне керівництво, 9-е видання. М.: Вільямс. 2015. 1376с.
9. Гамма, Р.Хелм, Р.Джонсон, Дж. Вліссідес, Прийоми об'єктно-орієнтованого проектування. Патерни проектування. С.-Пб: Питер, 2001. 368с.
10. Блінов, І.М., Романчик, В. С. Java. Методи програмування: навчально-методичний посібник. Минск : издательство «Четыре четверти», 2013. 896с.
11. Програмування на мові Java : веб-сайт. URL: <http://shtanyuk.tk/edu/nniit/java-new/html/11.html> (дата звернення 20.01.2021 р.)

12. Client-Server Application in Java. by Jasmine J. Ahuja, advisor Dr.Howard Blum. 1997. – 37.
13. Java ServerSocket : веб-сайт. URL: <https://www.javatpoint.com/java-serversocket-getremotesocketaddress-method> (дата звернення 05.02.2021 р.)
14. Java SE. ServerSocket. URL: <http://java-online.ru/java-socket.shtml> (дата звернення 10.03.2021 р.)
15. Wikipedia. James Goslong : веб-сайт. URL: <https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D1%81%D0%BB%D0%B8%D0%BD%D0%B3,%D0%94%D0%B6%D0%B5%D0%B9%D0%BC%D1%81> (дата звернення 20.12.2020 р.)
16. Історія мов програмування. Що допомогло Java увійти в кожен будинок : веб-сайт. URL: <https://habr.com/ru/post/306476/> (дата звернення 25.12.2020 р.)
17. Java Code Conventions. September 12, 1997. 24с. URL: <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf> (дата звернення 06.01.2021 р.)
18. JavaStudy. Spring MVC : веб-сайт. URL: <https://javastudy.ru/spring-mvc/spring-mvc-basic/> (дата звернення 12.03.2021 р.)
19. IntelliJ Idea. Навчальні відео : веб-сайт. URL: <https://www.jetbrains.com/ru-ru/idea/resources/> (дата звернення 20.12.2020 р.)
20. BetaCode. Java Basic : веб-сайт. URL: <https://betacode.net/10973/java-basic> (дата звернення 22.12.2020 р.)

ДОДАТОК А

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»
Факультет (інститут) Навчально-науковий інститут математики та
інформаційних технологій

(повна назва)
Кафедра Інформаційних технологій та систем

(повна назва)

Програма та методика тестування
на виконання програмної розробки (ПР):
Розробка клієнт-серверного Java-додатку для
автоматизації спілкування із клієнтами

Полтава – 2024

ЗМІСТ

ТЕСТ ПЛАН.....	36
1.Вступ.....	36
1.1.Мета.....	36
1.2. Вхідні дані.....	36
1.3. Мета тестування.....	36
2.Умови тестування.....	37
3.Стратегія процесу тестування.....	37
3.1. Тестування інтерфейсу.....	37
3.1.2. функціональне тестування.....	38
4. План робіт.....	38
5. Кінцеві результати.....	38
ТЕСТОВІ ВИПАДКИ	39

ТЕСТ-ПЛАН

1. Вступ

1.1. Мета

Мета даного тест плану - це опис клієнт-серверного додатку на мові програмування Java. Документ дозволяє отримати інформацію про план тестових робіт.

1.2 Вхідні дані

Клієнт-серверний додаток “Корпоративний чат”- написаний на платформі програмування Java.

1.3 Мета тестування

Метою перевірки клієнт-серверного додатку є перевірка коректної роботи всіх його елементів. Більшу частину тестування буде відведена тестування інтерфейсу, коректність логіки додатку, а також unit-тестам які будуть тестувати методи main, основних класів програми.

Підсумком процесу тестування будуть наступні матеріали:

- Висновок відносно загального стану, що надасть розробнику цього продукту картину відносно роботи програми.
- Звіт про результати тестування.
- Задokumentовані помилки при виявленні.

Тестування буде проводитися вручну, методом "неформального" тестування (ad-hoc testing) з позиції кінцевого користувача програми.

Таким чином, буде протестований як інтерфейс, так і основні можливості та функціонал програми.

Комп'ютерна система, що затверджена до перевірки:

Персональний комп'ютер із наступними характеристиками:

- Процесор Intel(R) Core(TM) i3-2120 CPU з потужністю 3.30 GHz
- 4,00 ГБ оперативної пам'яті
- Тип системи: 64-розрядна операційна система

- Операційна система Windows 8.1 x64 професійна

2. Умови тестування

Програма повинна правильно контролювати дії користувача. Система додатку система повинна правильно відповідати на введення користувача.

3. Стратегія процесу тестування

В процесі тестування клієнт-серверного додатку на мові програмування Java буде використано ad-hoc тестування там прості unit тести main методів, з огляду обмеженості ресурсів на формалізацію тестів.

Планується 2 етапи проведення процесу тестування:

- Перший етап полягає в складанні тест-плану.
- Другий етап буде присвячений тестуванню функціоналу серверної частини та клієнтської – перевірятиметься відклик системи на введення даних при запуску серверу у консолі, і введення даних при підключенні до серверу клієнта, який має здатність ввести дані, як з використанням інтерфейсу так і з використанням консолі.

Таким чином, буде протестований як інтерфейс клієнту, так і основні можливості програми клієнт-серверного додатку.

Коп'ютерна система і ПО, на якому буде проводитися тестування:

- 4,00 ГБ оперативної пам'яті
- ОС Windows 8.1 Професійна x64.
- Процесор Intel(R) Core(TM) i3-2120, 4 ядра по 3.30 GHz .

3.1. Тестування інтерфейсу

мета:

- Перевірка функціональності кнопки “Ok” клієнтського інтерфейсу при введенні адреси сервера;
- Перевірка функціональності кнопки “Ok” клієнтського інтерфейсу при введенні порту сервера;
- Перевірка функціональності кнопок “Ok” та “Cancel” клієнтського інтерфейсу при введенні ім'я, при підключенні до серверу;
- Перевірка кнопки “Ok” при введенні коректної інформації, при

підключенні до сервера;

- Перевірка функціональності кнопки "Exit" у вікні підключення до сервера;
- Перевірка функціональності запропонованих команд ботом у вікні чату;
- Перевірка функціональності введення повідомлень у вікні чату;

4. План робіт

Задача	Об'єм робіт	Дата початку	Дата закінчення
Укладання тест-плану	12 годин	01.05.2020	02.05.2021
Виконання тестування	12 години	02.05.2020	03.05.2021
Аналіз тестування	6 годин	03.05.2020	03.05.2021
Підведення підсумків	6 годин	03.05.2020	04.05.2021

5. Кінцеві результати

5.1 Підсумок

Підсумком проведення тестування є оформлений додаток «Тест-план» (цей документ) та «Тестові випадки», які містять результат процесу тестування.

ТЕСТОВІ ВИПАДКИ

Тестування інтерфейсу

Тестова задача: тестування запуску сервера і підключення клієнту до сервера.

1. Консоль введення порту сервера для його запуску(Рис.А.1).

```
"C:\Program Files\Java\jdk1.8.0_212\bin\java.exe" ...
```

Введіть порт сервера:

8080

Чат Сервер запущен.

```
Process finished with exit code -1
```

Рис.А.1. Консоль введення порту для запуску сервера

Перевірка підключення клієнта до сервера:

1. Вікно при підключенні клієнта та для введення адреси сервера працює нормально(Рис. А.2);

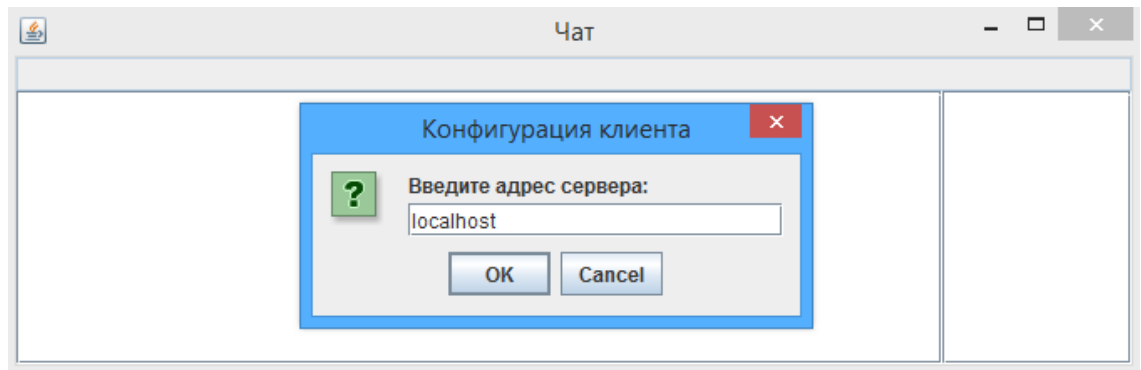


Рис. А.2. Вікно для введення адреси сервера

2. Кнопка «Ok» працює нормально й запускає наступне вікно для введення порту. (Рис. А.3);

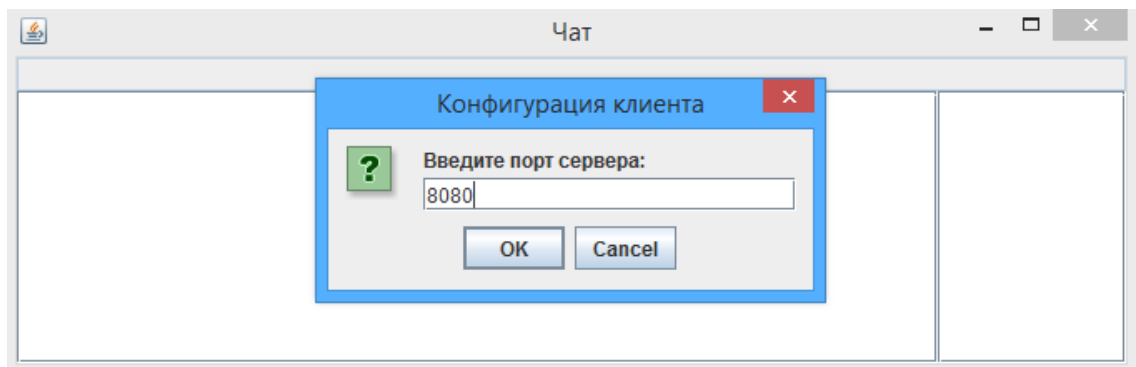


Рис. А.3. Вікно для введення порту сервера

3. Кнопка «Ok» працює нормально й запускає вікно для введення ім'я. (Рис. А.3);

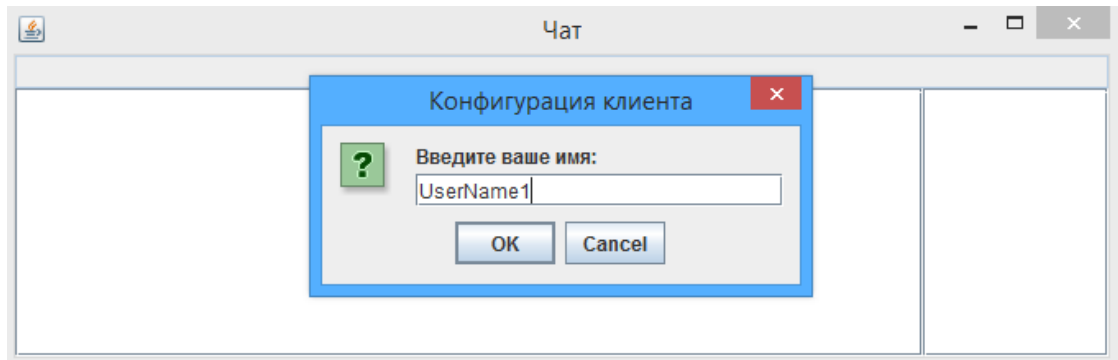


Рис.А. 3. Вікно для введення ім'я

4. Кнопка «Cancel» працює нормально та викликає вікно з повідомленням про не підключення до сервера(Рис. А.4.).

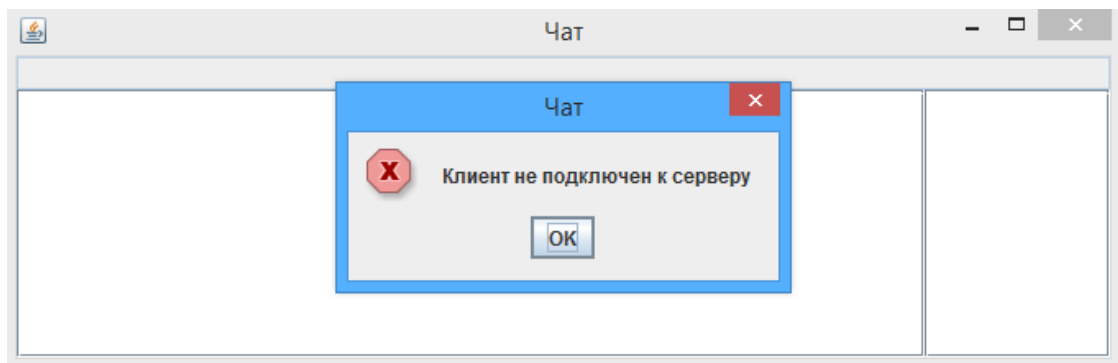


Рис.А. 4. Вікно з повідомленням про не підключення до сервера

5. Кнопка “Ok” після введення коректних даних при підключенні до сервера, викликає повідомлення про підключення до серверу(Рис А.5.), і далі після натискання знову кнопки “Ok” викликає вікно чату, працює нормально(Рис А.6.).

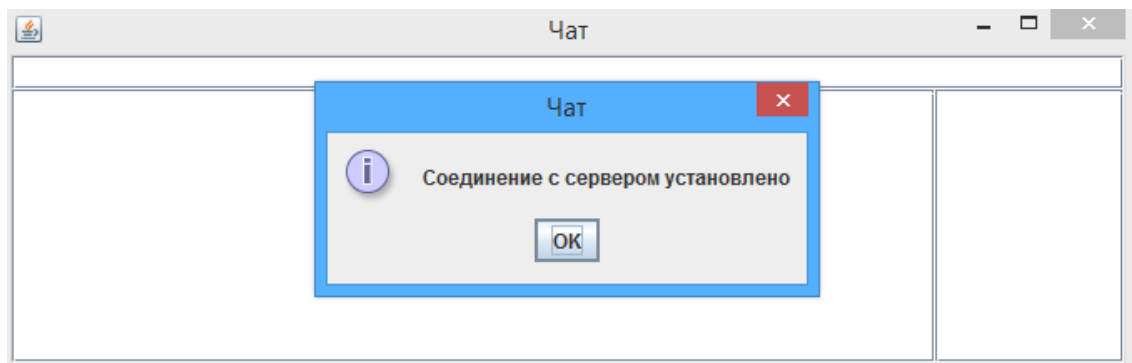


Рис.А. 5. Вікно з повідомленням про підключення до сервера

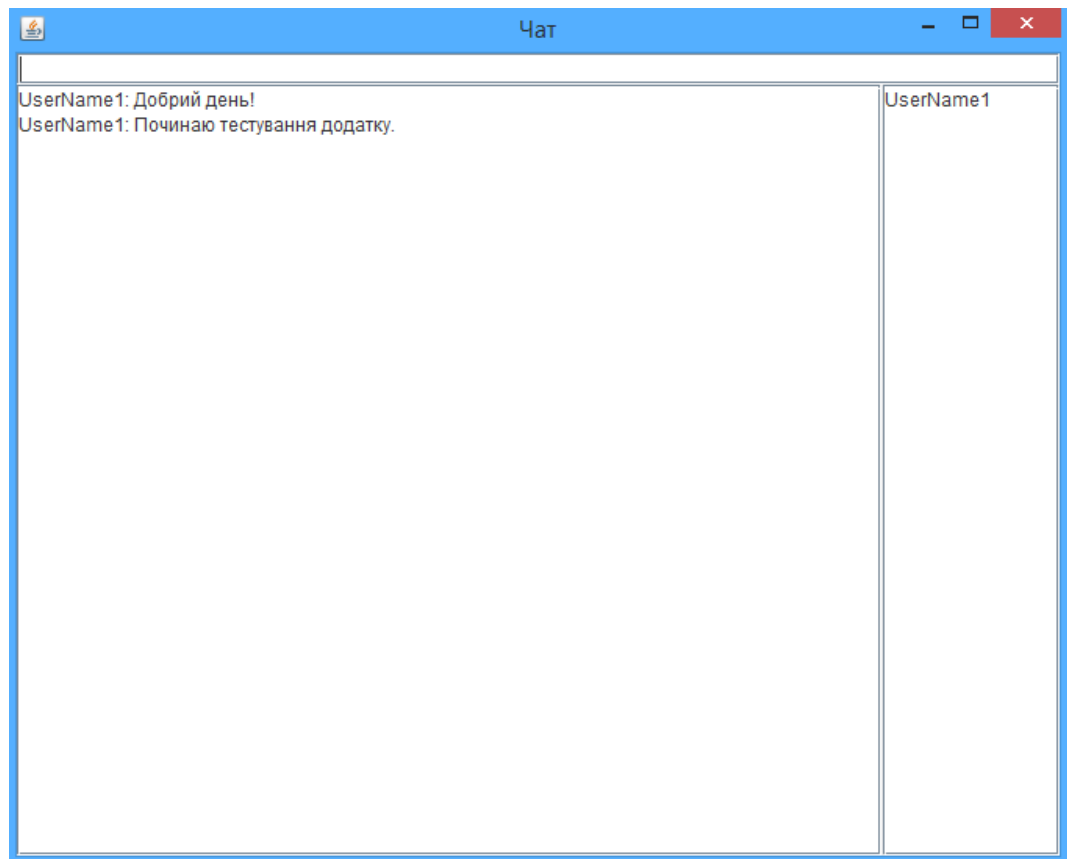


Рис.А. 6. Вікно чату після підключення клієнта

6. Кнопка "Exit" у вікні підключення до сервера, працює нормально(Рис. А.7).

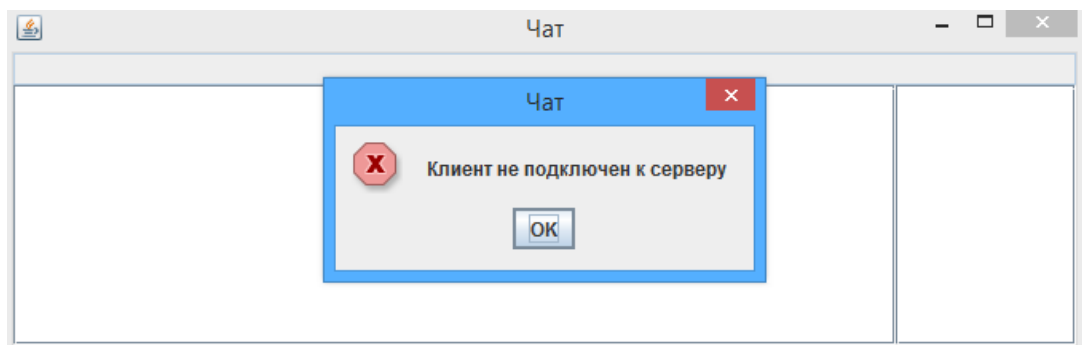


Рис.А. 7. Вікно після натискання кнопки "Exit" при введенні ім'я клієнта

7. Перевірка функціональності запропонованих команд ботом у вікні чату, працює нормально(Рис. А.8.);

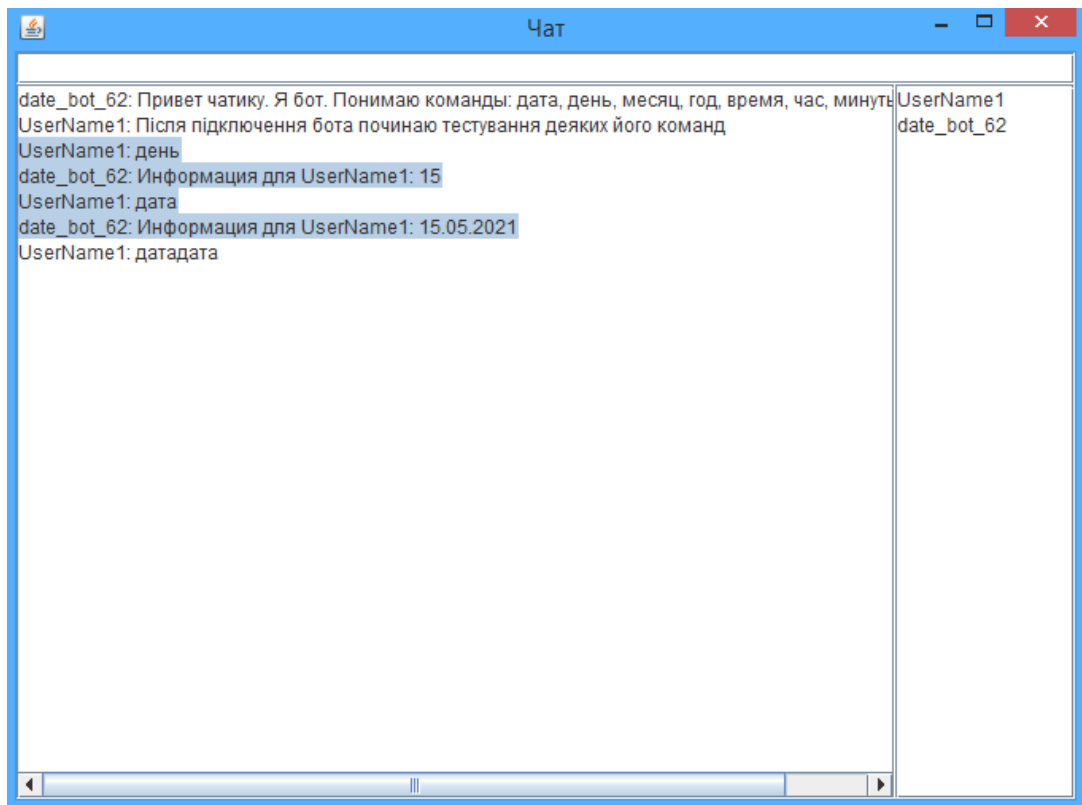


Рис. А. 9. Бот виконує команди користувача.

8. Введення повідомлень у вікні чату, працює нормально(Рис. А.10.);

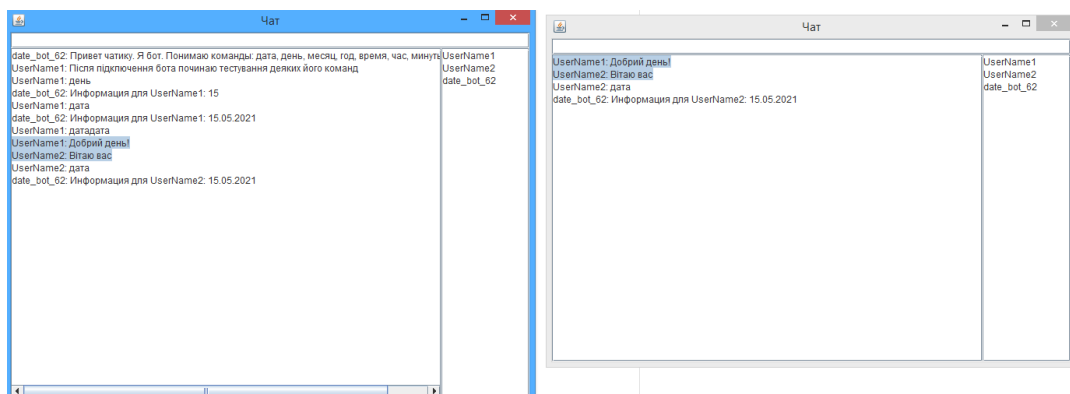


Рис. А. 10. Діалог між двома користувачами.

Результати тестувань клієнт-серверного додатку на платформі Java, на ПК (Windows 8.1) показали, що додаток відповідає усім встановленим вимогам.

ДОДАТОК В

Вихідний код класів які організовують роботу сервера – `Server.class`, `ConsoleHelper.class`, `Message.class`, `MessageType.enum`, `Connection.class`.

```
package Chat;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class Server {
    private static Map<String,Connection> connectionMap = new ConcurrentHashMap<>();
    public static void main(String[] args) {
        ConsoleHelper.writeMessage("Введіть порт сервера:");
        int port = ConsoleHelper.readInt();
        try(ServerSocket serverSocket = new ServerSocket(port)){
            ConsoleHelper.writeMessage("Чат Сервер запущено.");
            while (true){
                Socket socket = serverSocket.accept();
                new Handler(socket).start();
            }
        } catch (Exception exception){
            ConsoleHelper.writeMessage("Виникла помилка при запуску або роботі сервера.");
        }
    }
    private static class Handler extends Thread{
        Socket socket;
        public Handler(Socket socket) {
            this.socket = socket;
        }
        @Override
        public void run() {
```

```

        ConsoleHelper.writeMessage("Установлено новое соединение с " +
socket.getRemoteSocketAddress());
        String userName = null;
        try(Connection connection = new Connection(socket)){
            userName = serverHandshake(connection);
            sendBroadcastMessage(new Message(MessageType.USER_ADDED , userName));
            notifyUsers(connection, userName);
            serverMainLoop(connection , userName);
        }catch (IOException | ClassNotFoundException exception){
            ConsoleHelper.writeMessage("Ошибка при обмене данными с " +
socket.getRemoteSocketAddress());
        }
        if(userName != null){
            connectionMap.remove(userName);
            sendBroadcastMessage(new Message(MessageType.USER_REMOVED , userName));
        }
        ConsoleHelper.writeMessage("Соединение с " + socket.getRemoteSocketAddress() + "
закрыто.");
    }

    private String serverHandshake(Connection connection) throws IOException,
ClassNotFoundException {
        while (true) {
            connection.send(new Message(MessageType.NAME_REQUEST));
            Message message = connection.receive();
            if (message.getType() != MessageType.USER_NAME) {
                ConsoleHelper.writeMessage("Получено сообщение от " +
socket.getRemoteSocketAddress() + ". Тип повідомлення не відповідає протоколу.");
                continue;
            }
            String userName = message.getData();
            if (userName.isEmpty()) {
                ConsoleHelper.writeMessage("Спроба підключення до сервера з порожнім ім'ям
" + socket.getRemoteSocketAddress());
                continue;
            }

```

```

        if (connectionMap.containsKey(userName)) {
            ConsoleHelper.writeMessage("Попытка подключения к серверу с уже
используемым именем от " + socket.getRemoteSocketAddress());
            continue;
        }
        connectionMap.put(userName, connection);
        connection.send(new Message(MessageType.NAME_ACCEPTED));
        return userName;
    }
}

private void notifyUsers(Connection connection, String userName) throws IOException{
    for(String name : connectionMap.keySet()){
        if(name.equals(userName)){
            continue;
        }
        connection.send(new Message(MessageType.USER_ADDED, name));
    }
}

private void serverMainLoop(Connection connection, String userName)throws
IOException, ClassNotFoundException{
    while(true){
        Message message = connection.receive();
        if(message.getType() == MessageType.TEXT){
            String data = message.getData();
            sendBroadcastMessage(new Message(MessageType.TEXT, userName + ": " +
data));
        }else{
            ConsoleHelper.writeMessage("Отримано повідомлення від " +
socket.getRemoteSocketAddress() + ". Тип повідомлення не відповідає протоколу.");
        }
    }
}

public static void sendBroadcastMessage(Message message) {
    for (Connection connection : connectionMap.values()) {

```



```

        try{
            connection.send(message);
        }catch (IOException exception){
            ConsoleHelper.writeMessage("Не змогли надіслати повідомлення " +
connection.getRemoteSocketAddress());
        }
    }
}
}
}
}

```

```

package Chat;

```

```

import java.io.Closeable;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.SocketAddress;

```

```

public class Connection implements Closeable {
    private final Socket socket;
    private final ObjectOutputStream out;
    private final ObjectInputStream in;
    public Connection(Socket socket) throws IOException {
        this.socket = socket;
        this.out = new ObjectOutputStream(socket.getOutputStream());
        this.in = new ObjectInputStream(socket.getInputStream());
    }
    public void send(Message message)throws IOException{
        synchronized (out){
            out.writeObject(message);
        }
    }
    public Message receive() throws IOException, ClassNotFoundException {
        synchronized (in) {
            return (Message) in.readObject();
        }
    }
}

```

```

    }
}
public SocketAddress getRemoteSocketAddress(){
    return socket.getRemoteSocketAddress();
}
public void close()throws IOException{
    socket.close();
    out.close();
    in.close();
}
}

```

```
package Chat;
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```

public class ConsoleHelper {
    private static BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
    public static void writeMessage(String message) {
        System.out.println(message);
    }
    public static String readString() {
        while (true) {
            try {
                String inputString = reader.readLine();
                if (inputString != null) {
                    return inputString;
                }
            } catch (IOException exception) {
                writeMessage("Виникла помилка при спробі введення числа. Спробуйте ще раз.");
            }
        }
    }
}

```

```

    }
    public static int readInt() {
        while (true) {
            try {
                int inputNumber = Integer.parseInt(readString().trim());
                return inputNumber;
            } catch (NumberFormatException exception) {
                writeMessage("Виникла помилка при спробі введення числа. Спробуйте ще раз.");
            }
        }
    }
}

```

```

package Chat;

```

```

import java.io.Serializable;

```

```

public class Message implements Serializable {
    private final MessageType type;
    private final String data;
    public Message(MessageType type) {
        this.type = type;
        this.data = null;
    }
    public Message(MessageType type, String data) {
        this.type = type;
        this.data = data;
    }
    public MessageType getType() {
        return type;
    }
    public String getData() {
        return data;
    }
}

```

```
package Chat;
```

```
public enum MessageType {  
    NAME_REQUEST, USER_NAME, NAME_ACCEPTED, TEXT, USER_ADDED,  
    USER_REMOVED  
}
```

Вихідний код класів які організовують роботу клієнту та бот-клієнта –
ClientGuiController.class, Client.class, ClientGuiModel.class, ClientGuiView.class,
BotClient.class.

```
package Chat.client;
```

```
public class ClientGuiController extends Client {  
    private ClientGuiModel model = new ClientGuiModel();  
    private ClientGuiView view = new ClientGuiView(this);  
    @Override  
    protected SocketThread getSocketThread() {  
        return new GuiSocketThread();  
    }  
    public ClientGuiModel getModel() {  
        return model;  
    }  
    @Override  
    public void run() {  
        SocketThread socketThread = getSocketThread();  
        socketThread.run();  
    }  
    public static void main(String[] args) {  
        ClientGuiController client = new ClientGuiController();  
        client.run();  
    }  
    @Override  
    protected String getServerAddress() {
```

```

        return view.getServerAddress();
    }
    @Override
    protected int getServerPort() {
        return view.getServerPort();
    }
    @Override
    protected String getUserName() {
        return view.getUserName();
    }
    @Override
    protected void sendTextMessage(String text) {
        super.sendTextMessage(text);
    }
    public class GuiSocketThread extends SocketThread {
        @Override
        protected void processIncomingMessage(String message) {
            model.setNewMessage(message);
            view.refreshMessages();
        }
        @Override
        protected void informAboutAddingNewUser(String userName) {
            model.addUser(userName);
            view.refreshUsers();
        }
        @Override
        protected void informAboutDeletingNewUser(String userName) {
            model.deleteUser(userName);
            view.refreshUsers();
        }
        @Override
        protected void notifyConnectionStatusChanged(boolean clientConnected) {
            view.notifyConnectionStatusChanged(clientConnected);
        }
    }

```

```
}  
}
```

```
package Chat.client;
```

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;
```

```
public class ClientGuiView {  
    private final ClientGuiController controller;  
    private JFrame frame = new JFrame("Chat");  
    private JTextField textField = new JTextField(50);  
    private JTextArea messages = new JTextArea(10, 50);  
    private JTextArea users = new JTextArea(10, 10);  
  
    public ClientGuiView(ClientGuiController controller) {  
        this.controller = controller;  
        initView();  
    }  
    private void initView() {  
        textField.setEditable(false);  
        messages.setEditable(false);  
        users.setEditable(false);  
        frame.getContentPane().add(textField, BorderLayout.NORTH);  
        frame.getContentPane().add(new JScrollPane(messages), BorderLayout.WEST);  
        frame.getContentPane().add(new JScrollPane(users), BorderLayout.EAST);  
        frame.pack();  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        frame.setVisible(true);  
        textField.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                controller.sendTextMessage(textField.getText());  
                textField.setText("");  
            }  
        });  
    }  
}
```

```

    }
});
}

public String getServerAddress() {
    return JOptionPane.showInputDialog(
        frame,
        "Введите адрес сервера:",
        "Конфигурация клиента",
        JOptionPane.QUESTION_MESSAGE);
}

public int getServerPort() {
    while (true) {
        String port = JOptionPane.showInputDialog(
            frame,
            " Введіть порт сервера:",
            " Конфігурація клієнта ",
            JOptionPane.QUESTION_MESSAGE);
        try {
            return Integer.parseInt(port.trim());
        } catch (Exception e) {
            JOptionPane.showMessageDialog(
                frame,
                " Введено некоректний порт сервера. спробуйте ще раз.",
                " Конфігурація клієнта ",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}

public String getUserName() {
    return JOptionPane.showInputDialog(
        frame,
        "Введите ваше имя:",
        "Конфигурация клиента",
        JOptionPane.QUESTION_MESSAGE);
}

```

```

public void notifyConnectionStatusChanged(boolean clientConnected) {
    textField.setEditable(clientConnected);
    if (clientConnected) {
        JOptionPane.showMessageDialog(
            frame,
            " З'єднання з сервером встановлено ",
            "Чат",
            JOptionPane.INFORMATION_MESSAGE);
    } else {
        JOptionPane.showMessageDialog(
            frame,
            "Клиент не подключен к серверу",
            "Чат",
            JOptionPane.ERROR_MESSAGE);
    }
}

public void refreshMessages() {
    messages.append(controller.getModel().getNewMessage() + "\n");
}

public void refreshUsers() {
    ClientGuiModel model = controller.getModel();
    StringBuilder sb = new StringBuilder();
    for (String userName : model.getAllUserNames()) {
        sb.append(userName).append("\n");
    }
    users.setText(sb.toString());
}

}

package Chat.client;

import Chat.Connection;
import Chat.ConsoleHelper;
import Chat.Message;
import Chat.MessageType;

```



```
import java.io.IOException;
import java.net.Socket;

public class Client {
    protected Connection connection;
    private volatile boolean clientConnected = false;
    public static void main(String[] args) {
        Client client = new Client();
        client.run();
    }
    protected String getServerAddress() {
        ConsoleHelper.writeMessage("Введіть адресу сервера:");
        return ConsoleHelper.readString();
    }
    protected int getServerPort() {
        ConsoleHelper.writeMessage("Введіть порт сервера:");
        return ConsoleHelper.readInt();
    }
    protected String getUsername() {
        ConsoleHelper.writeMessage("Введіть ваше ім'я:");
        return ConsoleHelper.readString();
    }
    protected boolean shouldSendTextFromConsole() {
        return true;
    }
    public class SocketThread extends Thread {
        @Override
        public void run() {
            try {
                connection = new Connection(new Socket(getServerAddress(), getServerPort()));
                clientHandshake();
                clientMainLoop();
            } catch (IOException | ClassNotFoundException e) {
                notifyConnectionStatusChanged(false);
            }
        }
    }
}
```

```

    }
protected void clientHandshake() throws IOException, ClassNotFoundException {
    while (true) {
        Message message = connection.receive();
        if (message.getType() == MessageType.NAME_REQUEST) {
            String name = getUser_name();
            connection.send(new Message(MessageType.USER_NAME, name));
        } else if (message.getType() == MessageType.NAME_ACCEPTED) {
            notifyConnectionStatusChanged(true);
            return;
        } else {
            throw new IOException("Unexpected MessageType");
        }
    }
}

protected void clientMainLoop() throws IOException, ClassNotFoundException {
    while (true) {
        Message message = connection.receive();
        if (message.getType() == MessageType.TEXT) {
            processIncomingMessage(message.getData());
        } else if (MessageType.USER_ADDED == message.getType()) {
            informAboutAddingNewUser(message.getData());
        } else if (MessageType.USER_REMOVED == message.getType()) {
            informAboutDeletingNewUser(message.getData());
        } else {
            throw new IOException("Unexpected MessageType");
        }
    }
}

protected void processIncomingMessage(String message){
    ConsoleHelper.writeMessage(message);
}

protected void informAboutAddingNewUser(String userName){
    ConsoleHelper.writeMessage("Участник " + userName + " присоединился до чату.");
}

```

```

protected void informAboutDeletingNewUser(String userName){
    ConsoleHelper.writeMessage("Участник " + userName + " покинул чат");
}

protected void notifyConnectionStatusChanged(boolean clientConnected) {
    Client.this.clientConnected = clientConnected;
    synchronized (Client.this) {
        Client.this.notify();
    }
}

protected void sendTextMessage(String text) {
    try {
        connection.send(new Message(MessageType.TEXT, text));
    } catch (IOException exception) {
        ConsoleHelper.writeMessage("Неможливо відправити повідомлення ");
        clientConnected = false;
    }
}

protected SocketThread getSocketThread(){
    return new SocketThread();
}

public void run(){
    SocketThread socketThread = getSocketThread();
    socketThread.setDaemon(true);
    socketThread.start();
    try{
        synchronized (this){
            wait();
        }
    } catch (InterruptedException exception){
        ConsoleHelper.writeMessage("Виникла помилка під час роботи клієнта.");
        return;
    }
    while(clientConnected){
        String text = ConsoleHelper.readString();
    }
}

```

```

        if(text.equalsIgnoreCase("exit")){
            break;
        }
        if(shouldSendTextFromConsole()){
            sendTextMessage(text);
        }
    }
}

}

package Chat.client;

import java.util.Collections;
import java.util.Set;
import java.util.TreeSet;

public class ClientGuiModel {
    private final Set<String> allUserNames = new TreeSet<>();
    private String newMessage;
    public Set<String> getAllUserNames() {
        return Collections.unmodifiableSet(allUserNames);
    }
    public String getNewMessage() {
        return newMessage;
    }
    public void setNewMessage(String newMessage) {
        this.newMessage = newMessage;
    }
    public void addUser(String newUserName) {
        allUserNames.add(newUserName);
    }
    public void deleteUser(String userName) {
        allUserNames.remove(userName);
    }
}

```

```
package Chat.client;
```

```
import Chat.ConsoleHelper;
```

```
import java.io.IOException;
```

```
import java.text.SimpleDateFormat;
```

```
import java.util.Calendar;
```

```
public class BotClient extends Client {
```

```
    @Override
```

```
    protected SocketThread getSocketThread() {
```

```
        return new BotSocketThread();
```

```
    }
```

```
    @Override
```

```
    protected boolean shouldSendTextFromConsole() {
```

```
        return false;
```

```
    }
```

```
    @Override
```

```
    protected String getUserName() {
```

```
        return "date_bot_" + (int) (Math.random() * 100);
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        Client client = new BotClient();
```

```
        client.run();
```

```
    }
```

```
    public class BotSocketThread extends SocketThread {
```

```
        @Override
```

```
        protected void clientMainLoop() throws IOException, ClassNotFoundException {
```

```
            String hello = " Привіт чатику. Я робот. Розумію команди: дата, день, місяць, рік,  
час, година, хвилини, секунди.";
```

```
            BotClient.this.sendMessage(hello);
```

```
            super.clientMainLoop();
```

```
        }
```

```
        @Override
```

```
        protected void processIncomingMessage(String message) {
```

```
ConsoleHelper.writeMessage(message);
String userNameDelimiter = ": ";
String[] split = message.split(userNameDelimiter);
if (split.length != 2) return;
String messageWithoutUserName = split[1];
String format = null;
switch (messageWithoutUserName) {
    case "дата":
        format = "d.MM.YYYY";
        break;
    case "день":
        format = "d";
        break;
    case "місяць":
        format = "MMMM";
        break;
    case "год":
        format = "YYYY";
        break;
    case "время":
        format = "H:mm:ss";
        break;
    case "час":
        format = "H";
        break;
    case "хвилини":
        format = "m";
        break;
    case "секунди":
        format = "s";
        break;
}
if (format != null) {
    String answer = new
SimpleDateFormat(format).format(Calendar.getInstance().getTime());
```

```
        BotClient.this.sendMessage("Інформація для " + split[0] + ": " + answer);
    }
}
}
```