Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №9

З дисципліни «Технологія розробки програмного забезпечення»

Тема: «РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE»

Виконав:
студент 3 курсу,
групи ІА-12
Одемчук Н. О.

Перевірив:
ас. Колеснік В.М.

Київ 2023

Тема: РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE

Мета: Реалізувати взаємодію програми в одній з архітектур відповідно до обраної теми.

Завдання:

Клієнт для IRC-чатів з можливістю вказівки порту і адреси з'єднання, підтримка базових команд (підключення до чату, створення чату, установка імені, реєстрація, допомога і т.д.), отримання метаданих про канал.

Хід роботи

Для клієнта IRC (Internet Relay Chat) реалізована архітектура CLIENT-SERVER. Взаємодія клієнта та сервера встановлюється через такі компоненти:

1. Клас IRCClient:
    - Представляє додаток клієнта.
    - Обробляє загальні функції клієнта, включаючи підключення до сервера, відправку та отримання повідомлень і відключення.
    - Використовує екземпляр класу `IRCBridge` для взаємодії з сервером IRC.
2. Клас IRCBridge (абстрактний):
    - Служить абстрактним базовим класом, який визначає інтерфейс для мостів зв'язку.
    - Два конкретні виконання (`SocketBridge` та `WebSocketBridge`) розширюють цей клас, надаючи конкретні методи взаємодії.
3. Клас SocketBridge:
    - Конкретна реалізація класу `IRCBridge` з використанням TCP-сокету для зв'язку.
    - Обробляє методи, такі як `connect`, `disconnect`, `reconnect`, `send_data` та `receive_data`, що специфічні для зв'язку через сокет.
4. Клас WebSocketBridge:

- Заготовка для потенційної альтернативної реалізації за допомогою WebSocket для зв'язку.
- Зараз цей клас порожній і не реалізований.

5. Валідація конфігурації сервера:
   - Два класи (`BasicConfigValidation` та `AdvancedConfigValidation`) реалізують логіку перевірки конфігурації сервера.
   - Клас `IRCClient` використовує екземпляр одного з цих класів для перевірки деталей сервера перед встановленням з'єднання.
6. Клас MessageHandler:
   - Обробляє виконання команд IRC на основі введення користувача.
   - Відображає команди користувача на конкретні класи команд (наприклад, `JoinChannelCommand`, `LeaveChannelCommand`) та виконує їх.
7. Класи команд:
   - Класи, що представляють конкретні команди IRC (наприклад, join, part, names), які реалізують інтерфейс `Command`.
   - У кожному класі команди є метод `execute`, який виконує конкретну дію, пов'язану з командою.
8. Клас InputValidation:
   - Надає методи для перевірки введення користувача перед відправленням його на сервер.
   - Модифікує введення користувача на основі конкретних команд (наприклад, `/join`, `/part`, `/nick`).
9. Клас OutputValidation:
   - Перевіряє повідомлення, отримані від сервера, перед їхнім відображенням користувачу.
   - Додає мітку часу до отриманих даних для відображення.
10. Клас GUI:
    - Представляє графічний інтерфейс користувача за допомогою Tkinter.
    - Надає рамки для реєстрації (введення деталей сервера) та чату (відображення повідомлень, відправлення повідомлень).

- Взаємодіє з екземпляром `IRCClient` для обробки дій користувача

11. Основне виконання:
    - В блоку `__main__` створюється екземпляр `IRCClient`, і встановлюється перевірка конфігурації сервера.
    - Створюється екземпляр класу `GUI` з екземпляром `IRCClient`.
    - Клас `GUI` запускає головний цикл Tkinter.

Клієнт взаємодіє з сервером, підключаючись до нього за допомогою методу connect_to_server, надсилаючи повідомлення за допомогою send_message, отримуючи повідомлення за допомогою receive_data та відключаючись за допомогою disconnect_from_server. Клієнт взаємодіє з сервером через з'єднання за допомогою сокету, а користувач взаємодіє з клієнтом через графічний інтерфейс користувача (GUI).

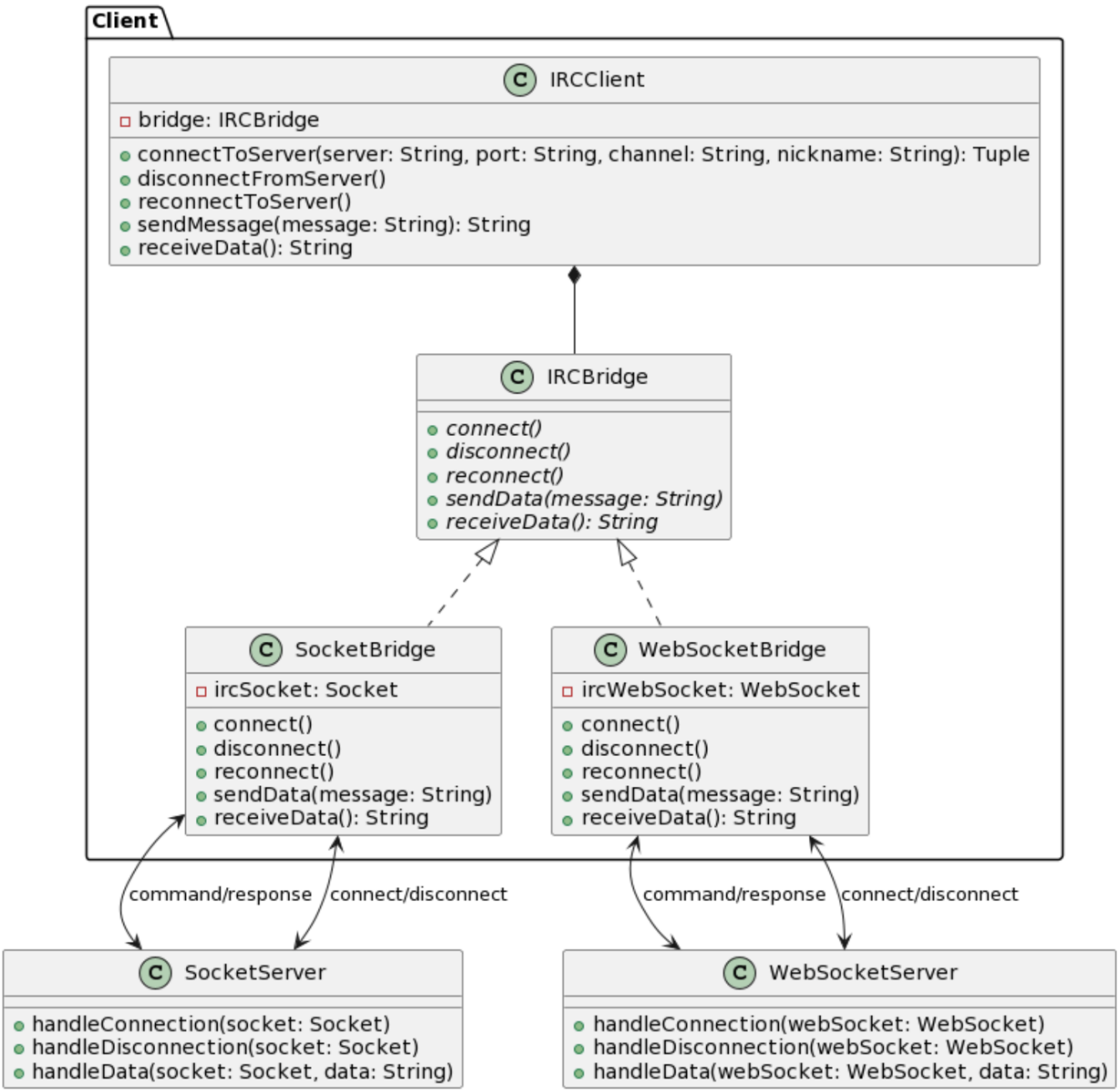

Рисунок 9.1 – Діаграма класів для архітектури CLIENT-SERVER

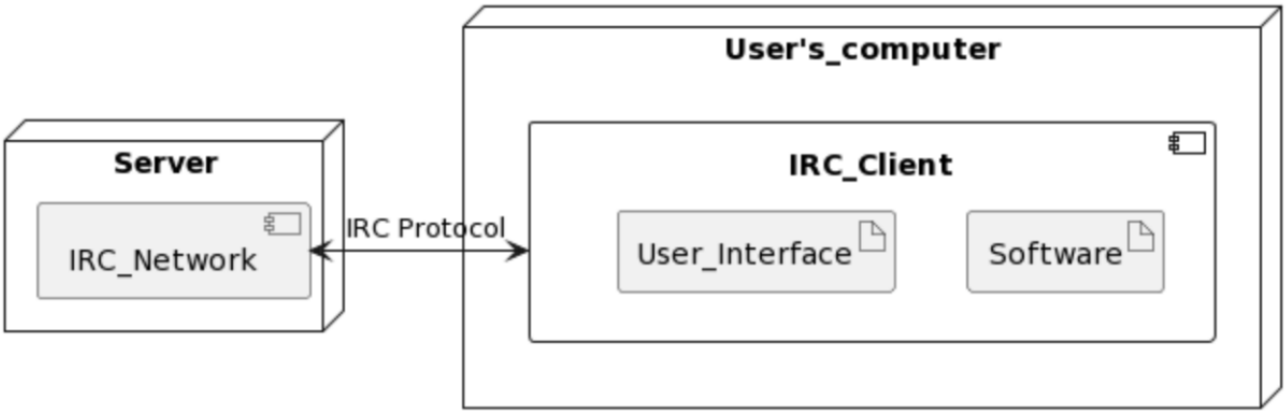

Рисунок 9.2 – Діаграма розгортання CLIENT-SERVER

Зараз хочу показати застосування CLIENT-SERVER архітектури для зберігання історії чату.

Мій код реалізує просту архітектуру клієнт-сервер для зберігання та отримання історії чату. Ось огляд того, як реалізована архітектура:

1. Серверна частина:
    - Клас `Server` відповідає за обробку вхідних підключень клієнтів та управління комунікацією з клієнтами.
    - Після прийняття підключення від клієнта створюється новий потік (`client_handler_thread`), який обробляє комунікацію з цим клієнтом незалежно.
    - Сервер безперервно очікує вхідних підключень у головному потоці.
2. Сервер історії:
    - Клас `HistoryServerHandler` відповідає за комунікацію з окремим сервером історії.
    - Створюється сокет сервера, і сервер безперервно очікує вхідних підключень.
    - Метод `handle_client` класу `Server` керує комунікацією з підключеним клієнтом. Він отримує повідомлення, оновлює історію чату

(`chat_history`) та надсилає оновлену історію клієнту.

3. Клієнтська сторона:
    - Клас `IRCClient` відповідає за управління клієнтською функціональністю.
    - Використовується екземпляр `HistoryServerHandler` для підключення до сервера історії, і клієнт безперервно надсилає та отримує історію чату від/до сервера.
    - Клієнт надсилає повідомлення на сервер за допомогою методу `send_message`. Перед відправленням повідомлення воно перевіряється за допомогою `InputValidation`.
    - Клієнт отримує та обробляє дані від сервера за допомогою методу `receive_data`. Отримані дані перевіряються за допомогою `OutputValidation`.
    - Клієнт також надсилає історію чату на сервер історії за допомогою методу `post_chat_history`.
    - Клієнт може запросити історію чату від сервера історії за допомогою методу `get_chat_history`.
4. Потокове виконання:
    - Використовуються потоки як на стороні сервера, так і на стороні клієнта для обробки одночасних підключень та безперервного отримання та обробки даних.

Загалом ця архітектура дозволяє зберігати та отримувати історію чату на сервері, а окремий сервер історії відповідає за обробку цих даних. Клієнт та сервер спілкуються за допомогою сокетів та обмінюються повідомленнями, що забезпечує просту, але ефективну модель клієнт-сервер для управління історією чату.

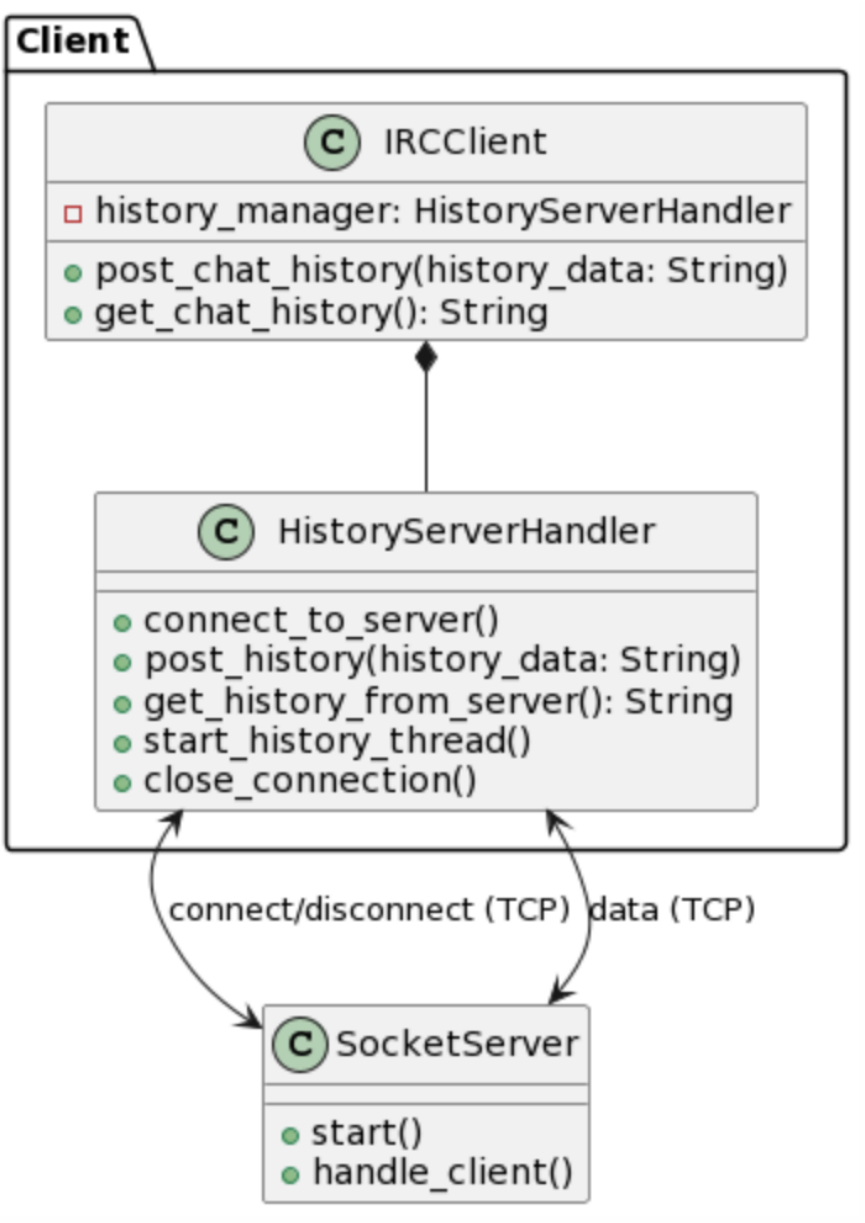

Рисунок 9.3 – Діаграма класів для архітектури CLIENT-SERVER для відображення історії чату

Висновки: У цій лабораторній роботі я реалізував архітектуру CLIENT-SERVER.