



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні системи та технологій

Лабораторна робота № 9

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Виконав

Студент групи IA-31:

Олея М. С.

Перевірив:

Мягкий М. Ю.

Київ 2025

Зміст

1. Мета:	3
2. Теоретичні відомості:.....	3
3. Завдання:.....	3
4. Хід роботи:.....	3
Призначення та доцільність:	4
Програмна реалізація:	6
5. Висновок	8
6. Контрольні питання:	8

1. Мета:

Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувати в проектованій системі одну із архітектур.

2. Теоретичні відомості:

Клієнт-серверна архітектура — модель, де клієнт відповідає за взаємодію з користувачем, а сервер — за зберігання та обробку даних. Тонкий клієнт (наприклад, вебзастосунки) передає більшість операцій на сервер, спрошууючи оновлення. Товстий клієнт (мобільні або десктопні програми) виконує логіку локально, зменшуючи навантаження на сервер і дозволяючи працювати офлайн. SPA (Single Page Application) — проміжний варіант: логіка на клієнті, але робота можлива лише з підключенням до сервера. Типова структура включає три рівні: клієнтський (інтерфейс), спільний (middleware) і серверний (бізнес-логіка та дані).

Peer-to-Peer (P2P) — децентралізована модель, де кожен вузол одночасно є клієнтом і сервером. Усі учасники рівноправні, обмінюються ресурсами без центрального сервера (наприклад, BitTorrent, блокчайн, Skype). Недоліки: складність забезпечення безпеки, синхронізації та пошуку даних у великих мережах.

Сервіс-орієнтована архітектура (SOA) — модульний підхід, де система складається з незалежних сервісів зі стандартизованими інтерфейсами (HTTP, SOAP, REST). Сервіси виконують конкретні бізнес-функції, обмінюються повідомленнями і можуть бути інтегровані через Enterprise Service Bus (ESB). SOA стала основою для мікросервісів.

Мікросервісна архітектура — створення додатків як набору незалежних малих сервісів, що взаємодіють через HTTP, WebSockets або AMQP. Кожен мікросервіс має власну логіку, життєвий цикл і може розгортатися автономно. Переваги: гнучкість, масштабованість і легке супроводження великих систем.

3. Завдання:

- 1) Ознайомитись з короткими теоретичними відомостями.
- 2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- 3) Реалізувати один з розглянутих шаблонів за обраною темою.
- 4) Реалізувати не менше 3-х класів відповідно до обраної теми.
- 5) Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

4. Хід роботи:

Варіант - 8

Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server)

Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна)

Призначення та доцільність:

У попередніх лабораторних роботах система "PowerShell Terminal" функціонувала як локальний монолітний додаток: інтерфейс користувача та логіка виконання команд знаходилися в одному процесі. Для реалізації можливості віддаленого керування (Remote Administration) було обрано архітектуру Клієнт-Сервер.

Цей підхід є доцільним, оскільки:

- Розподілення навантаження: Сервер може бути запущений на потужній віддаленій машині, а клієнт — на слабкому ноутбуці.
- Безпека та ізоляція: Клієнт не має прямого доступу до файлової системи сервера, він лише надсилає текстові команди.
- Масштабованість: До одного сервера в майбутньому можуть підключатися декілька клієнтів (адміністраторів).

Ми використали протокол TCP через клас TcpClient, оскільки він гарантує надійну доставку даних без втрат, що критично важливо для передачі тексту команд та їх результатів.

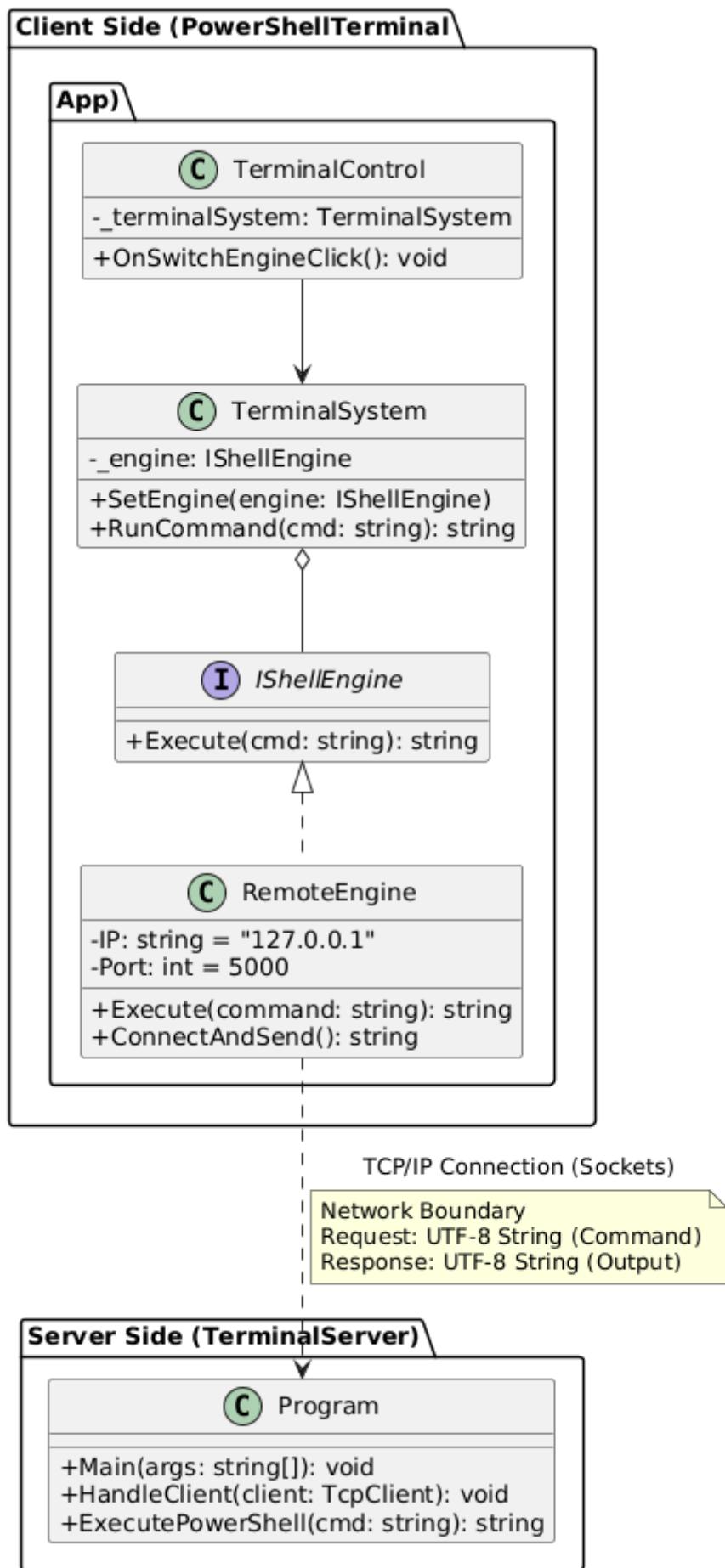


Рисунок 1 – Архітектура системи

Діаграма (рисунок 1) відображає розподілену архітектуру системи:

1. Клієнтська частина (Client Side):

- Інтегрована в існуючий додаток через патерн Bridge.
- Клас RemoteEngine реалізує інтерфейс IShellEngine. Для системи він виглядає як звичайний "двигун", але замість локального запуску процесу він відкриває мережеве з'єднання.

2. Мережева взаємодія:

- Зв'язок відбувається через сокети (TCP/IP) на порту 5000.
- Дані передаються у вигляді байтових масивів, кодованих у UTF-8.

3. Серверна частина (Server Side):

- Окремий консольний додаток (TerminalServer), який працює у нескінченному циклі, очікуючи підключень (TcpListener).
- При отриманні команди сервер запускає локальний процес PowerShell, перехоплює його вивід і відправляє назад клієнту.

Програмна реалізація:

a) Клієнтська частина - Забезпечення віддаленого доступу (RemoteEngine.cs):

```
public class RemoteEngine : IShellEngine
{
    private const string SERVER_IP = "127.0.0.1";
    private const int SERVER_PORT = 5000;

    public string GetEngineName() => "Remote Server (TCP)";

    public string Execute(string command)
    {
        try
        {
            using (TcpClient client = new TcpClient())
            {
                client.Connect(SERVER_IP, SERVER_PORT);
                NetworkStream stream = client.GetStream();

                byte[] data = Encoding.UTF8.GetBytes(command);
                stream.Write(data, 0, data.Length);

                byte[] buffer = new byte[8192];
                int bytesRead = stream.Read(buffer, 0, buffer.Length);

                string response = Encoding.UTF8.GetString(buffer, 0, bytesRead);
                return response;
            }
        }
        catch (SocketException)
```

```

        {
            return "[Network Error] Не вдалося підключитися до сервера. Запустіть
TerminalServer.exe!";
        }
        catch (Exception ex)
        {
            return $"[Client Error] {ex.Message}";
        }
    }
}

```

6) Серверна частина (Program.cs у проекті TerminalServer)

```

static void Main(string[] args)
{
    int port = 5000;
    TcpListener server = new TcpListener(IPAddress.Any, port);
    server.Start();

    Console.WriteLine($"[SERVER] Running on port {port}. Waiting for commands...");

    while (true)
    {
        try
        {
            TcpClient client = server.AcceptTcpClient();
            Console.WriteLine("[SERVER] Client connected!");

            Thread t = new Thread(() => HandleClient(client));
            t.Start();
        }
        catch (Exception ex)
        {
            Console.WriteLine($"[SERVER ERROR] {ex.Message}");
        }
    }
}

```

```

[SERVER] Running on port 5000. Waiting for commands...
[SERVER] Client connected!
[CMD] Received: dir

```

Рисунок 2 – Сервер запущено

```

PS User $ > dir

Каталог: C:\КПІ\Ворк\Misha\ТРПЗ\PowerShellTerminal

Mode          LastWriteTime    Length Name
----          -----          ----  --
d-----      20.12.2025     9:41   PowerShellTerminal.App
d-----      20.12.2025    11:56   TerminalServer
d-----      20.12.2025    11:35   Звіти
-a----      20.12.2025    11:53   2910 PowerShellTerminal.sln

```

Рисунок 3 – Комбінація з системною командою

Демонстрація роботи архітектури Клієнт-Сервер: Клієнт (графічний інтерфейс) надсилає команду `dir` через TCP-сокет, Сервер (консоль) отримує її, виконує та повертає результат

5. Висновок

У ході виконання лабораторної роботи було реалізовано взаємодію компонентів системи за архітектурою Клієнт-Сервер. Для цього було розроблено окремий серверний додаток (`TerminalServer`), що прослуховує TCP-порт та виконує отримані команди в середовищі `PowerShell`. Клієнтську частину було інтегровано в основний додаток через патерн `Bridge` (клас `RemoteEngine`), що дозволило перемикатися між локальним та віддаленим виконанням команд без зміни інтерфейсу користувача. Використання сокетів (`TcpClient`, `TcpListener`) забезпечило надійний двосторонній обмін даними. Реалізація підтвердила гнучкість обраної архітектури та можливість масштабування системи для віддаленого адміністрування.

6. Контрольні питання:

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель організації комп’ютерних систем, де клієнти (зазвичай користувацькі програми або пристрой) роблять запити на сервер, який обробляє ці запити і повертає результати. Клієнт відповідає за інтерфейс користувача та ініціацію запитів, сервер — за обробку даних, зберігання та логіку.

2. Розкажіть про сервіс-орієнтовану архітектуру (SOA).

SOA — це архітектурний підхід, де функціональність програми реалізована у вигляді сервісів — автономних компонентів, що виконують конкретні

бізнесзавдання. Кожен сервіс має чіткий інтерфейс і взаємодіє з іншими через стандартизовані протоколи (наприклад, HTTP, SOAP, REST).

3. Якими принципами керується SOA?

Основні принципи SOA:

- Автономність сервісів — сервіси працюють незалежно.
- Стандартизовані інтерфейси — сервіси взаємодіють через визначені API.
- Повторне використання — сервіси можна використовувати в різних системах.
- Легко інтегрувати — сервіси повинні легко поєднуватись у складні процеси.
- Слабке зв'язування (loose coupling) — зміни в одному сервісі мінімально впливають на інші.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють через стандартизовані повідомлення або API, наприклад через SOAP або REST. Кожен сервіс публікує свій інтерфейс (WSDL, OpenAPI), і інші сервіси можуть викликати його методи, обмінюючись даними у формі XML, JSON або інших форматах.

5. Як розробники дізнаються про існуючі сервіси і як робити до них запити?

- Реєстр сервісів (Service Registry) — централізована база, де зареєстровані всі сервіси та їхні інтерфейси.
- Документація API (наприклад OpenAPI/Swagger).
- Запити здійснюються через стандартні протоколи (HTTP, SOAP, REST) за адресою сервісу і з використанням описаних методів та форматів даних.

5. У чому полягають переваги та недоліки клієнт-серверної моделі?

Переваги:

- Центральне зберігання даних, легший контроль безпеки.
- Легко масштабувати сервери.
- Клієнти можуть бути простими, вся логіка на сервері.

Недоліки:

- Сервер може стати вузьким місцем (single point of failure).
- Високі вимоги до потужності сервера при великій кількості клієнтів.
- Залежність клієнта від сервера — без доступу до сервера система не працює.

7. У чому полягають переваги та недоліки однорангової (peer-to-peer) моделі взаємодії? Переваги:

- Відсутність централізованого сервера, підвищена стійкість до відмов.
- Можливість прямого обміну ресурсами між учасниками.
- Масштабування «горизонтальне» — додаючи вузли, підвищуєш потужність.

Недоліки:

- Складніше забезпечувати безпеку та контроль доступу.
- Кожен вузол відповідає за управління ресурсами.
- Важче координувати оновлення і синхронізацію даних.

8. Що таке мікросервісна архітектура? Мікросервісна архітектура — це підхід, де додаток складається з малих, незалежних сервісів, кожен з яких реалізує одну бізнес-функцію і може розгортатися окремо. Вона є розвитком SOA, але з більш дрібними і автономними компонентами.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

- HTTP/HTTPS + REST
- gRPC
- SOAP
- Message brokers: RabbitMQ, Kafka, MQTT для асинхронної взаємодії
- WebSockets для реального часу

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, якщо у проєкті між веб-контролерами та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

Це не повноцінна SOA, а локальне використання сервісів всередині одного додатку. SOA передбачає автономні, незалежні сервіси, доступні через стандартизовані протоколи для інших систем.