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 «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи»

«2. HTTP–сервер»

Виконав:
студент групи – ІА–32
Непотачев Іван Дмитрович

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Взаємодія компонентів системи.

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

Тема проєкту: HTTP–сервер (state, builder, factory method, mediator, composite, p2p) Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах.

Зміст

Теоретичні відомості

Peer-to-Peer архітектура
Peer-to-Peer (P2P) архітектура – це модель мережевої взаємодії, в якій кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. На відміну від клієнт-серверної моделі, де є чітке розділення на клієнти й сервери, P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері.
Основними принципами P2P-архітектури є:
• Децентралізація – відсутність центрального сервера, що зменшує залежність від одного вузла, підвищуючи стійкість мережі до збоїв і атак.
• Рівноправність вузлів – кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси).
• Розподіл ресурсів – вузли надають доступ до своїх власних ресурсів, таких як обчислювальна потужність, дисковий простір або файли.

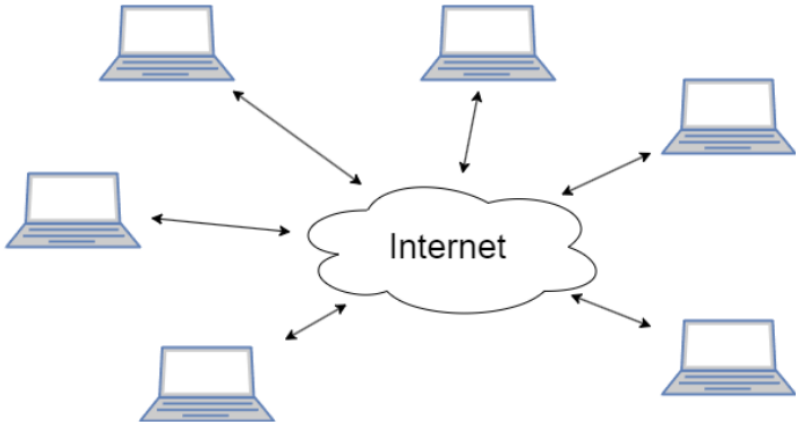

Рисунок 1.1. Peer-to-Peer архітектура

Основними сферами де peer-to-peer архітектура знайшла широке застосування є файлообмінники (BitTorrent), криптовалюти та інші блокчейн-технології, інтернет телефонія та відеоконференції (Skype, Zoom), розподілені обчислення (SETI@home, BOINC). До основних проблемних зон можна віднести безпеку, синхронізацію даних та пошук ресурсів. Через централізацію складно контролювати дані, які передаються. Ефективність пошуку даних знижується зі збільшенням кількості вузлів у мережі і для підвищеня ефективності пошуку потрібно застосовувати спеціальні алгоритми.

Хід роботи

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.

• Реалізувати взаємодію розподілених частин:

– Для клієнт-серверних варіантів: реалізація клієнтської і серверної частини додатків, а також загальної частини (middleware); зв'язок клієнтської і серверної частин за допомогою WCF, TcpClient, .NET-Remoting на розсуд виконавця.

– Для однорангових мереж: реалізація взаємодії клієнтських додатків за допомогою WCF Peer to peer channel.

– Для SOA додатків: реалізація сервісу, що надає послуги клієнтським застосуванням; викладання сервісу в хмару або підняття у вигляді Web Service на локальній машині; використання токенів для передачі даних про автентифікації, двостороннє шифрування.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє спроєктовану архітектуру.
Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури.

Реалізація Peer-to-Peer архітектури

Для реалізації HTTP-сервера використано Peer-to-Peer архітектуру, де кожен вузол мережі одночасно виступає як клієнт і як сервер, здійснюючи прямий обмін даними між собою без централізованого сервера. Кожен PeerNode працює як незалежний HTTP-сервер, здатний приймати запити від інших вузлів, а також ініціювати запити до них через PeerClient.

Архітектура P2P реалізована у класах PeerNode, PeerClient та HttpServer, які забезпечують повноцінну функціональність кожного вузла мережі. Такий підхід є особливо ефективним для розподілених систем, де необхідно забезпечити високу доступність та відмовостійкість.

```java
public class PeerNode {  5 usages
    private final String name;  3 usages
    private final HttpServer server;  5 usages
    private final PeerClient client;  2 usages
    private final Statistics stats;  3 usages
    private final List<String> peers;  2 usages

    public PeerNode(String name, int port, List<String> peers) {  2 usages
        this.name = name;
        this.stats = new Statistics();
        RequestHandler handler = new RequestHandler();
        this.server = new HttpServer(port,  mediator: null);
        ServerMediator mediator = new ConcreteServerMediator(server, handler, stats);
        this.client = new PeerClient();
        this.peers = peers;
    }

    public void start() {  2 usages
        System.out.println(name + " starting on port " + server.getPort());
        server.Start();
    }

    public void stop() {  2 usages
        server.Stop();
    }

    public void broadcastRequest(String url) {  2 usages
        for (String peerUrl : peers) {
            try {
                System.out.println(name + " sending request to " + peerUrl);
                String response = client.sendGet( url: peerUrl + url);
                System.out.println("Response from " + peerUrl + ":\n" + response);
            } catch (IOException e) {
                System.err.println("Peer " + peerUrl + " unreachable.");
            }
        }
    }

    public int getTotalRequests() {  no usages
        return stats.getTotalRequests();
    }
}
```

Рис. 2.1 – Код класу PeerNode

```java
package p2p;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class PeerClient {  2 usages

    public String sendGet(String url) throws IOException {  1 usage
        String host = url.split( regex: ":")[0];
        int port = Integer.parseInt(url.split( regex: ":")[1].split( regex: "/")[0]);
        String path = "/" + url.split( regex: "/",  limit: 2)[1];

        try (Socket socket = new Socket(host, port);
             PrintWriter out = new PrintWriter(socket.getOutputStream(),  autoFlush: true);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

            out.println("GET " + path + " HTTP/1.1");
            out.println("Host: " + host);
            out.println("Connection: close");
            out.println();
            out.flush();

            StringBuilder response = new StringBuilder();
            String line;
            while ((line = in.readLine()) != null) {
                response.append(line).append("\n");
            }
            return response.toString();
        }
    }
}
```

Рис. 2.2 – Код класу PeerClient

Застосування цієї архітектури забезпечує:

1. Децентралізовану комунікацію: Кожен PeerNode може самостійно приймати HTTP-запити через свій HttpServer та ініціювати запити до інших вузлів через PeerClient. Це усуває єдину точку відмови, характерну для клієнт-серверної архітектури.
2. Автономність вузлів: Кожен вузол має повний стек обробки запитів, включаючи стан сервера, медіатор для координації та фабрики для генерації відповідей.
3. Масштабованість: Додавання нових вузлів до мережі не вимагає змін в існуючій архітектурі. Новий PeerNode просто отримує список відомих peer'ів і може почати взаємодію:
4. Відмовостійкість: При недоступності одного вузла інші можуть продовжувати роботу та взаємодіюти між собою. Клас PeerClient обробляє помилки зв'язку, не впливаючи на стабільність вузла-сервера.
5. Ефективне використання ресурсів: Кожен вузол обробляє власні запити та може розподіляти навантаження між іншими доступними peer'ами через механізм broadcast.

У нашому випадку P2P архітектура дозволяє створити мережу взаємопов'язаних HTTP-серверів, де кожен вузол може:

- Приймати запити від клієнтів та інших вузлів
- Ініціювати запити до інших вузлів мережі
- Автономно функціонувати навіть при часткових збоях мережі
- Динамічно додавати та видаляти зв'язки з іншими вузлами

Зображення архітектури Peer-to-Peer

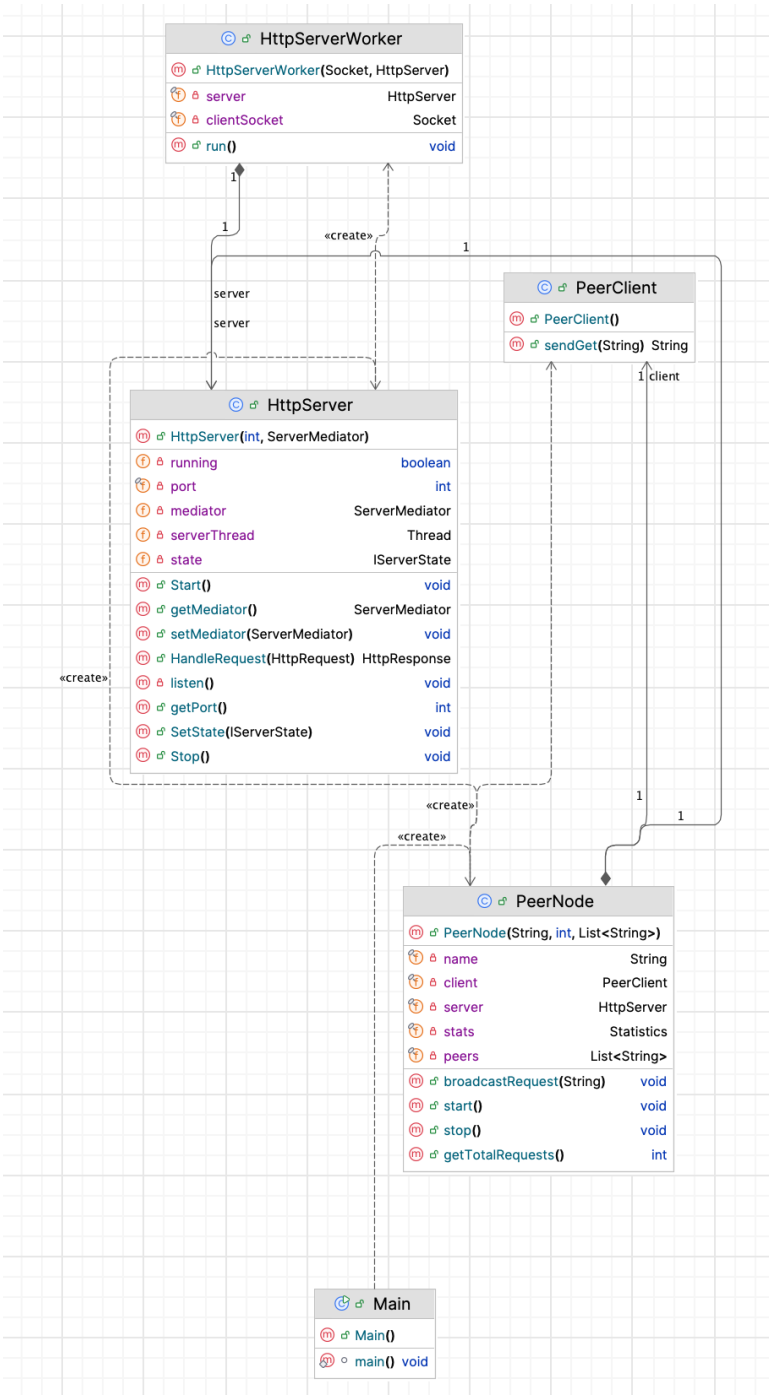

Рис. 3.1 – Структура архітектури Peer-to-Peer

Структура реалізації:

- PeerNode – основний клас вузла, що інтегрує HttpServer та PeerClient
- PeerClient – відповідає за ініціювання HTTP-запитів до інших вузлів

- HttpServer – обробляє вхідні HTTP-запити (модернізований для роботи в P2P мережі)

- HttpServerWorker – обробляє окремі підключення в окремих потоках

Посилання на репозиторій: https://github.com/IvanGodPro24/trpz
Посилання на звіти: https://github.com/IvanGodPro24/trpz_reports

Висновки

Висновки: під час виконання лабораторної роботи, було реалізовано та досліджено Peer-to-Peer архітектуру для HTTP-сервера, що дозволило створити децентралізовану мережу взаємопов'язаних вузлів. Кожен PeerNode був реалізований як автономна одиниця, здатна одночасно виконувати функції сервера (приймання запитів) та клієнта (ініціювання запитів до інших вузлів). Це усунуло єдину точку відмови, характерну для традиційної клієнт-серверної архітектури, та забезпечило високу доступність системи. P2P архітектура ефективно поєдналася з раніше реалізованими патернами проектування. Клас PeerNode інтегрував ці патерни в єдину систему, демонструючи гнучкість та масштабованість архітектури. Загалом, архітектура продемонструвала свою ефективність для розподілених систем, де важливі надійність, масштабованість та рівномірне розподілення навантаження між учасниками мережі.

Питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура – це модель, у якій клієнт надсилає запит до сервера, а сервер обробляє запит і повертає результат. Тобто клієнт – ініціатор запиту, сервер – постачальник ресурсу.

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA) – це підхід, при якому система складається з незалежних сервісів, що взаємодіють через стандартизовані інтерфейси та протоколи. Кожен сервіс виконує певну бізнес-функцію й може бути використаний повторно в різних застосунках.

3. Якими принципами керується SOA?

Основні принципи SOA:

- Слабке зв’язування (Loose coupling) – сервіси мінімально залежать один від одного.
- Повторне використання (Reusability) – сервіси можна використовувати повторно.
- Інтероперабельність – взаємодія через стандартизовані протоколи (HTTP, SOAP, REST).

- Відкриті інтерфейси – сервіси доступні через опис.

- Масштабованість і керованість.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють через мережу за допомогою повідомлень і стандартних протоколів (SOAP, REST, HTTP, XML, JSON).
Зазвичай комунікація відбувається через сервісну шину (ESB) або API-шлюзи, які координують обмін даними.

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

- Інформація про сервіси зберігається у реєстрі сервісів (Service Registry).

- Для пошуку використовується UDDI або документація API (наприклад Swagger, OpenAPI).

- Запити надсилаються через HTTP-запити або виклики SOAP/REST API.

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

Переваги:
- Централізоване керування даними.

- Вища безпека та контроль доступу.

- Простота оновлення та адміністрування.

Недоліки:
- Залежність від сервера (його збій блокує систему).

- Може бути перевантаження сервера при великій кількості клієнтів.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

Переваги:

- Відсутність центрального сервера.

- Висока стійкість до збоїв.
- Добра масштабованість.

Недоліки:

- Складність забезпечення безпеки та узгодження даних.
- Важко контролювати доступ і автентифікацію.

8. Що таке мікро-сервісна архітектура?

Мікросервісна архітектура – це стиль розробки, у якому система складається з невеликих незалежних сервісів, кожен з яких виконує окрему бізнес-функцію і може розгортатися, оновлюватися й масштабуватися окремо. Це еволюція SOA з фокусом на легкість, автономність і контейнеризацію.

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

- HTTP / HTTPS (REST API)
- gRPC (ефективний двійковий протокол Google)
- AMQP, MQTT, Kafka – для асинхронної комунікації через черги повідомлень.
- WebSocket – для двостороннього зв’язку в реальному часі.

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

Ні, тому що такий підхід – це шарова архітектура (Layered Architecture), а не повноцінна SOA. У SOA сервіси – це окремі, незалежно розгорнуті компоненти, які можуть взаємодіяти через мережу та бути використані іншими системами. У шаровій архітектурі сервіси – лише частина внутрішньої логіки програми.