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»

Виконала:

студентка групи - ІА-34
Мартинюк Т.В.

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

**Тема**: Взаємодія компонентів системи.

**Мета**: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented 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.

## Теоретичні відомості:

### Архітектура Client-Server

Архітектура Client-Server є однією з найпоширеніших моделей взаємодії додатків, де система поділена на дві основні частини: клієнт і сервер. Клієнт — це частина, яка ініціює запити, наприклад, веб-браузер або мобільний додаток, що надсилає команди на сервер для отримання даних або виконання дій. Сервер, у свою чергу, обробляє ці запити, зберігає дані, забезпечує логіку бізнесу та повертає відповіді. Ця модель централізована, що полегшує управління, безпеку та масштабування сервера, але має вразливості, такі як точка відмови (якщо сервер падає, вся система зупиняється) і потенційне перевантаження сервера при великому трафіку.

### Архітектура Peer-to-Peer

Архітектура Peer-to-Peer (P2P) передбачає децентралізовану взаємодію, де кожен вузол (пір) у мережі діє як рівноправний учасник, виконуючи ролі і клієнта, і сервера одночасно. Замість центрального сервера, піри безпосередньо обмінюються даними, ресурсами чи повідомленнями один з одним, що робить систему стійкою до відмов (якщо один пір виходить, інші продовжують працювати) і добре масштабованою для великих мереж. Однак, P2P має виклики з безпекою (складніше контролювати доступ), пошуком пірів (потрібні механізми

discovery, як DHT чи ручне введення IP) і ефективністю в мережах з NAT/фаєрволами.

**Архітектура Service-Oriented Architecture (SOA)**

Архітектура Service-Oriented Architecture (SOA) базується на ідеї поділу системи на незалежні сервіси, які взаємодіють через стандартизовані інтерфейси (наприклад, SOAP, REST або повідомлення). Кожен сервіс виконує конкретну функцію (наприклад, аутентифікація, платежі, зберігання даних) і може бути розроблений окремо, що сприяє модульності, повторному використанню коду та легкому масштабуванню. SOA є слабко зв'язаною, дозволяючи сервісам працювати на різних платформах чи мовах, але вимагає управління оркестрацією (наприклад, через ESB — Enterprise Service Bus) і може мати проблеми з продуктивністю через мережеві виклики.

**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, .NETRemoting 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.

**Хід роботи:**

У проєктованій системі було реалізовано архітектуру Peer-to-Peer для взаємодії між екземплярами сервера. Кожен сервер діє як пір, дозволяючи прямому обміну даними через роути /p2p/to (надсилання запиту іншому піру) та /p2p/ping (відповідь на запит). Це забезпечує децентралізовану мережу, де сервери можуть запитувати статус або статистику один в одного без центрального вузла, використовуючи стандартний HTTP-протокол для простоти та інтеграції з існуючою архітектурою.

**Результати запитів для піру на порті 8083**

http://localhost:8083/p2p/ping

```
P2P Ping: OK
IP: 192.168.50.150
Port: 8083
Time: 2025-12-13T02:22:24.589338
```

Рисунок 1 - Локальна відповідь піра на порту 8083

http://localhost:8083/p2p/to?target=192.168.50.150:8082

```
P2P запит до 192.168.50.150:8082/p2p/ping

Статус: 200

P2P Ping: OK
IP: 192.168.50.150
Port: 8082
Time: 2025-12-13T02:21:48.534656100
Total requests handled: 0
```

Рисунок 2 - P2P-запит з порту 8083 до порту 8082

**Результати запитів для піру на порті 8082**

http://localhost:8082/p2p/ping

```
P2P Ping: OK
IP: 192.168.50.150
Port: 8082
Time: 2025-12-13T02:23:04.666217300
Total requests handled: 0
```

Рисунок 3 - Локальна відповідь піра на порту 8082

http://localhost:8082/p2p/to?target=127.0.0.1:8083

```
P2P запит до 127.0.0.1:8083/p2p/ping

Статус: 200

P2P Ping: OK
IP: 192.168.50.150
Port: 8083
Time: 2025-12-13T02:23:54.257894200
Total requests handled: 0
```

Рисунок 4 - P2P-запит з порту 8082 до порту 8083

Лістинг коду:

```
public class P2PForwardRoute implements HttpRoute {

    @Override
    public HttpResponse execute(HttpRequest request) {
        String target = request.getQueryParams().get("target");
        String path = request.getQueryParams().getOrDefault("path", "/p2p/ping");

        if (target == null || target.isEmpty()) {
            Map<String, String> headers = new HashMap<>();
            headers.put("Content-Type", "text/plain; charset=UTF-8");

            String body = "Помилка 400: параметр порт обов'язковий";

            return new HttpResponse("400 Bad Request", headers, body);
        }
```

```
        try {
            URL url = new URL("http://" + target + path);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");
            conn.setConnectTimeout(5000);
            conn.setReadTimeout(5000);

            int statusCode = conn.getResponseCode();

            StringBuilder responseBody = new StringBuilder();
            if (statusCode >= 200 && statusCode < 300) {
                try (BufferedReader reader = new BufferedReader(
                        new InputStreamReader(conn.getInputStream()))) {
                    String line;
                    while ((line = reader.readLine()) != null) {
                        responseBody.append(line).append("\n");
                    }
                }
            } else {
                responseBody.append("HTTP error: ").append(statusCode)
                        .append(" ").append(conn.getResponseMessage());
            }

            String html = "<pre>P2P запит до " + target + path + "\n\n" +
                    "Статус: " + statusCode + "\n\n" +
                    responseBody.toString() + "</pre>";

            Map<String, String> headers = new HashMap<>();
            headers.put("Content-Type", "text/html; charset=UTF-8");

            return HttpResponseDirector.Ok(html, headers);

        } catch (Exception e) {
            String error = "Пір недоступний: " + target + "\nПомилка: " + e.getMessage();
            return HttpResponseDirector.ServiceUnavailable(error, null);
        }
    }
}
```

```
package routes.p2p;

import database.repos.ServerStatisticsService;
import http.HttpRequest;
import http.HttpResponse;
import http.HttpResponseDirector;
import http.HttpRoute;
import serverConfigs.ConfigHandler;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;
```

```
public class P2PPingRoute implements HttpRoute {

    private final int serverPort;

    public P2PPingRoute(int serverPort) {
        this.serverPort = serverPort;
    }

    @Override
    public HttpResponse execute(HttpRequest request) {
        try {
            String myIp = InetAddress.getLocalHost().getHostAddress();

            String body = "P2P Ping: OK\n" +
                    "IP: " + myIp + "\n" +
                    "Port: " + serverPort + "\n" +
                    "Time: " + LocalDateTime.now() + "\n";
            Map<String, String> headers = new HashMap<>();
            headers.put("Content-Type", "text/plain; charset=UTF-8");

            return HttpResponseDirector.Ok(body, headers);

        } catch (UnknownHostException e) {
            return HttpResponseDirector.InternalServerError("Cannot determine local IP", null);
        }
    }
}
```

**Висновок**: У ході лабораторної роботи було реалізовано архітектуру Peer-to-Peer у системі багатопотокового HTTP-сервера. Кожен екземпляр сервера виступає як рівноправний пір, здатний приймати запити та надсилати їх іншим пірам безпосередньо по HTTP без центрального посередника. Такий підхід ідеально підходить до проєкту, оскільки сервер уже є повноцінним HTTP-вузлом з роутингом і обробкою запитів, що дозволяє природно інтегрувати P2P-взаємодію без введення додаткових протоколів чи залежностей. Це забезпечує децентралізовану взаємодію, стійкість до відмов окремих вузлів та простоту масштабування мережі.

**Контрольні питання:**

1. **Що таке клієнт-серверна архітектура?**

Клієнт-серверна архітектура – це модель взаємодії додатків, у якій система поділена на дві основні ролі: клієнт (ініціює запити, наприклад, браузер або мобільний додаток) і сервер (обробляє запити, зберігає дані та виконує бізнес-логіку). Клієнти надсилають запити серверу, а сервер повертає відповіді. Це централізована модель, де сервер є основним постачальником ресурсів.

2. **Розкажіть про сервіс-орієнтовану архітектуру.**

Сервіс-орієнтована архітектура (SOA) – це підхід до проєктування систем, у якому функціональність поділена на незалежні сервіси, що взаємодіють через стандартизовані інтерфейси. Сервіси є автономними, повторно використовуваними компонентами, які виконують конкретні бізнес-функції. SOA сприяє модульності, гнучкості та інтеграції різнорідних систем.

3. **Якими принципами керується SOA?**

SOA базується на таких ключових принципах: стандартизовані контракти сервісів, слабка зв’язаність (loose coupling), абстрагування реалізації, повторне використання сервісів, автономність, бездержавність (statelessness), виявлення сервісів (discoverability), композиція сервісів та інтероперабельність.

4. **Як між собою взаємодіють сервіси в SOA?**

Сервіси в SOA взаємодіють через мережеві виклики за допомогою стандартизованих протоколів (наприклад, SOAP, REST, XML-RPC або повідомлень). Запити та відповіді передаються у форматі XML або JSON, часто з використанням ESB (Enterprise Service Bus) для оркестрації, маршрутизації та трансформації повідомлень.

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

Розробники дізнаються про сервіси через реєстр сервісів (наприклад, UDDI або сучасні API Gateway, Swagger/OpenAPI). Контракти сервісів описуються у WSDL (для SOAP) або OpenAPI (для REST), що містить інформацію про ендпоінти, методи, параметри та формати даних. Запити робляться через HTTP-клієнти або спеціальні бібліотеки, дотримуючись описаного контракту.

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

Переваги: централізоване управління даними та логікою, легка безпека, простота масштабування сервера, чіткий поділ ролей.

Недоліки: сервер є єдиною точкою відмови, можливе перевантаження при великій кількості клієнтів, залежність клієнтів від доступності сервера, проблеми з масштабуванням при швидкому зростанні.

7. **У чому полягають переваги та недоліки одноранговової моделі взаємодії?**

Переваги: децентралізованість (немає єдиної точки відмови), висока стійкість і масштабування (кожен пір додає ресурси), ефективне використання ресурсів мережі.

Недоліки: складність пошуку пірів і координації, проблеми з безпекою (складніше контролювати доступ), вищі вимоги до NAT-траверсингу та пропускної здатності, потенційна нестабільність через вихід пірів з мережі.

8. **Що таке мікро-сервісна архітектура?**

Мікросервісна архітектура – це еволюція SOA, у якій додаток поділено на невеликі, незалежні сервіси, кожен з яких відповідає за одну бізнес-функцію, має власну базу даних і розгортається окремо. Сервіси спілкуються через легковагові протоколи (зазвичай REST або повідомлення) і можуть бути розроблені різними командами на різних технологіях.

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

Найпоширеніші протоколи: HTTP/REST з JSON (для синхронної взаємодії), gRPC (для високопродуктивного RPC), повідомлення через брокери (Kafka,

RabbitMQ, NATS) для асинхронної взаємодії, а також GraphQL для гнучких запитів. Формати даних — переважно JSON або Protobuf.

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, це не є повноцінною SOA. Такий підхід – це класична тришарова архітектура (presentation – business – data access) у монолітному додатку, де "сервіси" є внутрішніми компонентами без мережевої взаємодії. SOA вимагає незалежних, автономних сервісів, що взаємодіють через мережу за стандартизованими інтерфейсами, а не локальних класів у одному додатку.