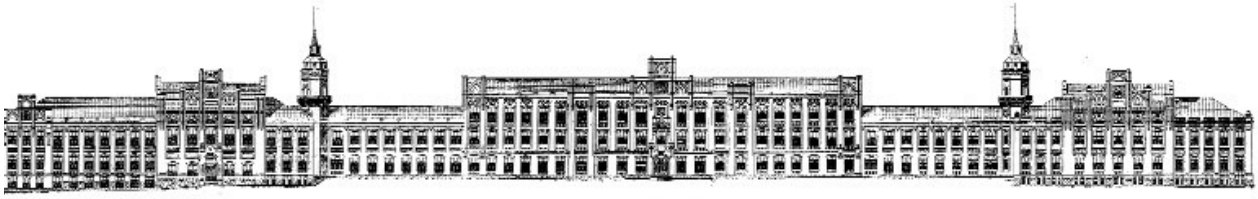

Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №9

з дисципліни «Технології розроблення

програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Виконала
студентка групи ІА-32:
Павлюк В. О.

Зміст

Table of Contents

Вступ

Мета цієї лабораторної це вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

Як об’єкт проєктування обрано тему «Web crawler»: програмний сканер, що розпізнає структуру сторінок сайту, переходить за посиланнями, збирає інформацію за заданим терміном, видаляє несемантичні елементи (рекламу, скрипти тощо), зберігає очищені дані у вигляді структурованого набору HTML-файлів і веде статистику відвідань та метадані.

1 Теоретичні відомості

1.1. Клієнт-серверна архітектура

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти. Тонкий клієнт – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки.

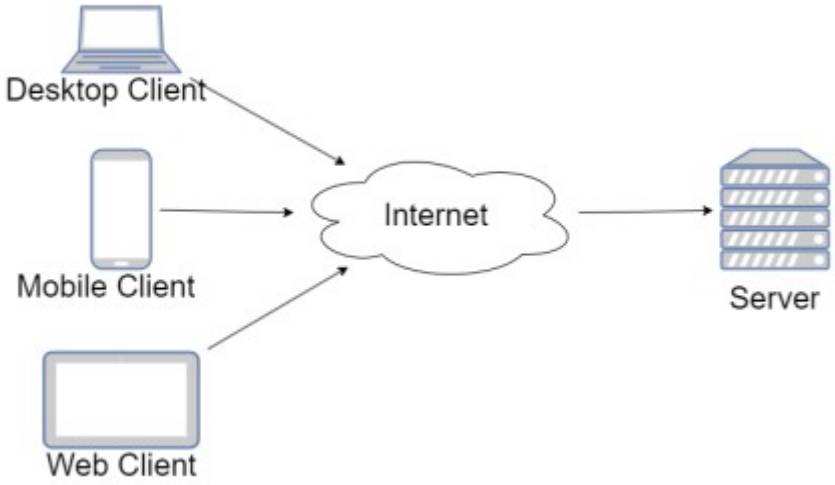

Рисунок 1.1. Клієнт-серверна архітектура

1.2. 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.

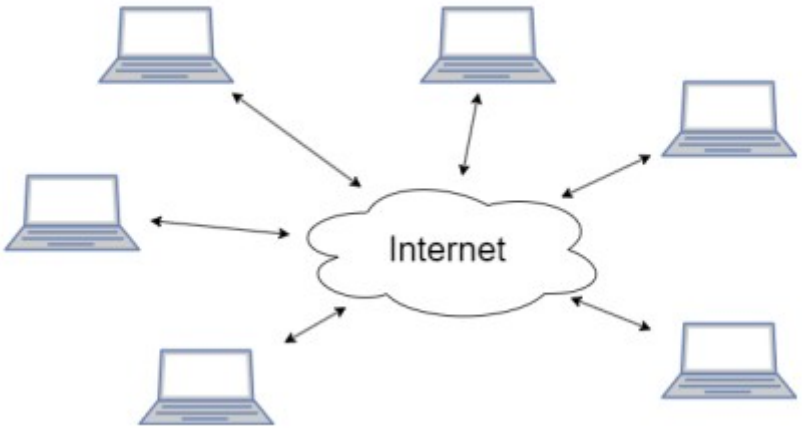

Рисунок 1.2. Peer-to-Peer архітектура

1.3. Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура (SOA, англ. service-oriented architecture) – модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов’язаних (англ. Loose coupling) сервісів або служб, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами [11]. Історично сервіс-орієнтована архітектура появилася як альтернатива монолітній архітектурі, в які вся система розроблялася та розгорталася як одне ціле. Програмні комплекси, розроблені відповідно до сервіс-орієнтованою архітектурою, зазвичай реалізуються як набір веб-служб (або веб-сервісів), які, як правило, взаємодіють по HTTP з використанням SOAP або REST. Ці служби надають певні бізнес-функції, наприклад, отримання інформації про наявність матеріалів на складі. Сервіси взаємодіють між собою тільки за рахунок обміну повідомленнями, без створення спеціальних інтеграцій для доступу до однієї інформації, наприклад, до однієї бази даних.

1.4. Мікро-сервісна архітектура.

Сама назва дає зрозуміти, що мікро-сервісна архітектура є підходом до створення серверного додатку як набору малих служб [11]. Це означає, що архітектура мікро-сервісів головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи

AMQP. Кожен мікросервіс реалізує специфічні можливості в предметній області і свою бізнес-логіку в рамках конкретного обмеженого контексту, повинна розроблятись автономно і розвертатись незалежно

Визначення мікросервісів із книги Іраклі Надарейшвілі, Ронні Мітра, Метта Маклarті та Майка Амундсена (О'Рейлі) «Архітектура мікросервісів»: «Мікросервіс – це компонент із чітко визначеними межами, який можна розгортати незалежно, і підтримує взаємодію за допомогою зв'язку на основі повідомлень. Архітектура мікросервісів – це стиль розробки високоавтоматизованих систем програмного забезпечення, що легко розвивати та яке складається з мікросервісів, орієнтованих на певні можливості». Мікросервіси забезпечують чудові можливості супроводження в величезних комплексних системах з високою масштабуємістю за рахунок створення додатків, заснованих на множині незалежно розгортуючих служб з автономними життєвими циклами.

2 Хід роботи

2.1. Реалізація P2P-взаємодії: вузли crawler обмінюються знайденими URL через мережу та додають їх у власний планувальник.

Завдання

- Реалізувати TCP peer-сервер/клієнт для обміну повідомленнями.
- Інтегрувати P2P у шаблонний метод через хуки.
- Забезпечити потокобезпечне додавання отриманих URL у UrlScheduler.

2.2 Діаграма

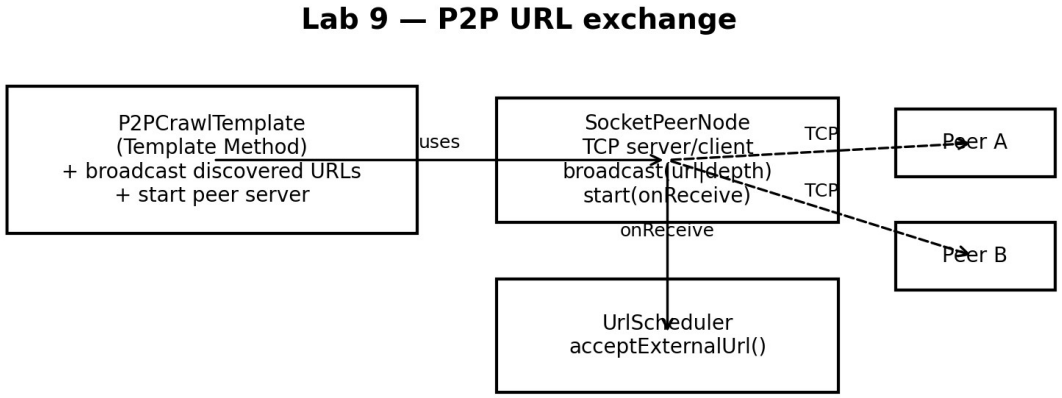

Рисунок 2.2 - Діаграма

2.3. Фрагмент коду (Java)

SocketPeerNode: старт сервера та обробка з’єднань

```
   public void start(BiConsumer<String, Integer> onUrlReceived) throws Exception
{
      if (running) return;
      running = true;
      serverSocket = new ServerSocket(localPort);
      serverSocket.setSoTimeout(1000);

      acceptThread = new Thread(() -> {
         while (running) {
            try {
```

```
                Socket socket = serverSocket.accept();
                handleConnection(socket, onUrlReceived);
            } catch (SocketTimeoutException e) {
                // ignore, loop again
            } catch (Exception e) {
                if (running) {
                    System.out.println("Peer server error: " + e.getMessage());
                }
            }
        }
    }, "p2p-accept-" + localPort);
    acceptThread.setDaemon(true);
    acceptThread.start();
    System.out.println("P2P peer started on port " + localPort + ", peers=" + peers);
  }

  private void handleConnection(Socket socket, BiConsumer<String, Integer>
onUrlReceived) {
    Thread t = new Thread(() -> {
      try (Socket s = socket;
           BufferedReader in = new BufferedReader(new
InputStreamReader(s.getInputStream()))) {
        String line;
        while ((line = in.readLine()) != null) {
          String[] parts = line.split("\\|");
          if (parts.length != 2) continue;
          String url = parts[0];
          int depth;
          try {
            depth = Integer.parseInt(parts[1]);
          } catch (Exception e) {
            continue;
          }
          onUrlReceived.accept(url, depth);
        }
      } catch (Exception ignored) {
      }
    }, "p2p-conn");
    t.setDaemon(true);
    t.start();
  }

  public void broadcast(String url, int depth) {
    String msg = url + "|" + depth;
```

```
        for (PeerEndpoint peer : peers) {
            try (Socket socket = new Socket(peer.getHost(), peer.getPort());
                 BufferedWriter out = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()))) {
                out.write(msg);
                out.newLine();
                out.flush();
            } catch (Exception e) {
                // peers may be offline; ignore to keep crawling
            }
        }
    }

    public void stop() {
        running = false;
        try {
            if (serverSocket != null) serverSocket.close();
        } catch (Exception ignored) {}
        try {
            if (acceptThread != null) acceptThread.join(1500);
```

P2PCrawlTemplate: запуск peer та broadcast знайдених URL

```
package ua.kpi.webcrawler.template;

import ua.kpi.webcrawler.core.DefaultContentProcessorFactory;
import ua.kpi.webcrawler.core.UrlScheduler;
import ua.kpi.webcrawler.http.HttpClient;
import ua.kpi.webcrawler.memento.CrawlCheckpointStore;
import ua.kpi.webcrawler.model.CrawlProfile;
import ua.kpi.webcrawler.model.ScanSession;
import ua.kpi.webcrawler.p2p.SocketPeerNode;
import ua.kpi.webcrawler.repository.InMemoryPageRepository;
import ua.kpi.webcrawler.repository.InMemorySessionRepository;

import java.util.List;

public class P2PCrawlTemplate extends DefaultCrawlTemplate {

    private final SocketPeerNode peerNode;

    public P2PCrawlTemplate(HttpClient httpClient,
                            DefaultContentProcessorFactory processorFactory,
                            InMemorySessionRepository sessionRepo,
                            InMemoryPageRepository pageRepo,
```

```
                        CrawlCheckpointStore checkpointStore,
                        SocketPeerNode peerNode) {
        super(httpClient, processorFactory, sessionRepo, pageRepo, checkpointStore);
        this.peerNode = peerNode;
    }

    @Override
    protected void beforeCrawl(CrawlProfile profile, ScanSession session,
UrlScheduler scheduler) throws Exception {
        if (peerNode != null) {
            peerNode.start((url, depth) -> {
                scheduler.acceptExternalUrl(url, depth);
                System.out.println("Received URL from peer: " + url + " depth=" + depth);
            });
        }
    }

    @Override
    protected void onLinksDiscovered(CrawlProfile profile, ScanSession session,
List<String> links, int depthForLinks) {
        if (peerNode == null) return;
        for (String link : links) {
            peerNode.broadcast(link, depthForLinks);
        }
    }

    @Override
    protected void afterCrawl(CrawlProfile profile, ScanSession session, UrlScheduler
scheduler) throws Exception {
        if (peerNode != null) {
            peerNode.stop();
        }
    }
}
```

Висновки

У ході лабораторної роботи було реалізовано реальний P2P-обмін URL через TCP-сокети. Інтеграція виконана через Template Method (хуки), а планувальник URL підтримує прийом зовнішніх URL і коректну роботу з кількома потоками.

Контрольні запитання

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель побудови програмних систем, у якій клієнт надсилає запити, а сервер обробляє ці запити та повертає відповіді. Клієнт відповідає за інтерфейс і взаємодію з користувачем, а сервер — за обробку даних, бізнес-логіку та доступ до ресурсів.

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA) — це підхід до побудови програмних систем, у якому функціональність реалізується у вигляді незалежних сервісів. Кожен сервіс надає чітко визначений інтерфейс і може використовуватися іншими компонентами або системами.

3. Якими принципами керується SOA?

SOA базується на таких принципах: слабка зв’язаність сервісів, чітко визначені контракти (інтерфейси), повторне використання сервісів, автономність сервісів, можливість їхнього повторного розгортання та незалежність від конкретної технології реалізації.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси в SOA взаємодіють через стандартизовані інтерфейси, використовуючи повідомлення. Обмін даними зазвичай відбувається через мережу за допомогою протоколів HTTP, SOAP або REST, що забезпечує платформну незалежність.

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

Інформація про сервіси зберігається у сервісних реєстрах або документації (наприклад, WSDL для SOAP або OpenAPI для REST). Запити до сервісів виконуються через публічні API з використанням стандартних протоколів і форматів даних.

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

Перевагами клієнт-серверної моделі є централізоване керування даними, зручність адміністрування та безпеки. Недоліками є залежність від сервера, можливість перевантаження та складність масштабування при великій кількості клієнтів.

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

Перевагами однорангової (peer-to-peer) моделі є відсутність центрального вузла, висока масштабованість і стійкість до відмов. Недоліками є складність забезпечення безпеки, контролю та узгодженості даних між вузлами.

8. Що таке мікросервісна архітектура?

Мікросервісна архітектура — це різновид сервіс-орієнтованої архітектури, у якому система складається з невеликих незалежних сервісів, кожен з яких відповідає за окрему бізнес-функцію і може розгортатися та масштабуватися незалежно.

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

Найчастіше використовуються HTTP/HTTPS з REST або gRPC, а також асинхронні протоколи й брокери повідомлень, такі як AMQP, MQTT, Kafka. Формати даних зазвичай включають JSON, XML або Protocol Buffers.

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

Ні, такий підхід сам по собі не є сервіс-орієнтованою архітектурою. Це є багатошарова (layered) архітектура з виділеним сервісним шаром. SOA передбачає наявність автономних сервісів із чіткими контрактами та мережевою взаємодією, а не лише логічний поділ коду всередині одного застосунку.