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»

«Web crawler»

Виконала
студентка групи – ІА–31
Горлач Дар’я Дмитрівна

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: Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p) Веб–сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

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.

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.

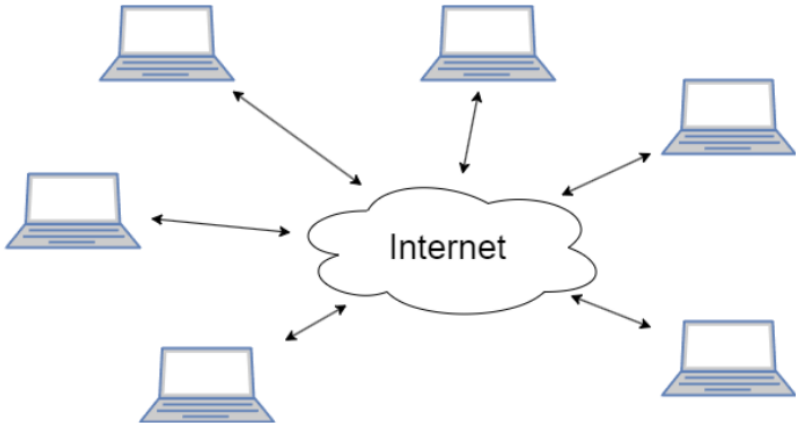

Рисунок 1. Peer-to-Peer архітектура

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

Реалізація архітектури

Для реалізації Web Crawler використано Peer-to-Peer архітектуру, оскільки вона забезпечує розподілену, децентралізовану обробку веб-сторінок, де кожен вузол мережі одночасно є і краулером, і сховищем даних. Це дозволяє створювати масштабовані, відмовостійкі системи збору веб-даних без єдиної точки відмови.

P2P архітектура реалізована у класах Peer, PeerNode та інтегрована з існуючим веб-краулером, що дозволяє кожному вузлу не лише завантажувати та обробляти сторінки, але й обмінюватися результатами з іншими учасниками мережі. Такий підхід є особливо ефективним для розподілених веб-краулерів, де необхідно масштабувати обробку великих обсягів даних.

```
3 usages   1 implementation
public interface Peer {
    3 usages   1 implementation
    String getId();

    2 usages   1 implementation
    void connect(Peer peer);

    1 usage   1 implementation
    void sendHtml(String url, String html);

    no usages   1 implementation
    void receiveHtml(String url, String html);
}
```

Рис. 2 – Код інтерфейсу Peer

```java
public class PeerNode implements Peer {

    5 usages
    private final String id;
    4 usages
    private final Map<String, String> sharedHtmlStorage = new HashMap<>();

    2 usages
    public PeerNode(String id) {
        this.id = id;
    }

    3 usages
    @Override
    public String getId() {
        return id;
    }

    2 usages
    @Override
    public void connect(Peer peer) {
        System.out.println("[" + id + "] підключився до peer " + peer.getId());
    }

    1 usage
    @Override
    public void sendHtml(String url, String html) {
        System.out.println("[" + id + "] надсилає HTML для " + url + ": " + html);
        sharedHtmlStorage.put(url, html);
    }
```

Рис. 3 – Код класу PeerNode

```java
public class WebCrawlerApp {
    dariahorlach *
    public static void main(String[] args) {
        PeerNode peer1 = new PeerNode( id: "Peer-1");
        PeerNode peer2 = new PeerNode( id: "Peer-2");

        peer1.connect(peer2);
        peer2.connect(peer1);

        PageFetcher realFetcher = new HttpPageFetcher();
        PageFetcherProxy proxy = new PageFetcherProxy(realFetcher);

        PageHistory history = new PageHistory();

        ScriptCleanerHandler scriptHandler = new ScriptCleanerHandler(history);
        AdsCleanerHandler adsHandler = new AdsCleanerHandler(history);
        KeywordSearchHandler keywordHandler = new KeywordSearchHandler(history);

        scriptHandler.setNext(adsHandler);
            adsHandler.setNext(keywordHandler);

        SimpleWebCrawler crawler1 =
                new SimpleWebCrawler(proxy, scriptHandler, peer1);

        crawler1.crawl( url: "https://example.com");

        history.printHistory();
    }
}
```

Рис. 4 – Код класу WebCrawlerApp

Використання цієї архітектури надає:

1. Децентралізоване зберігання даних: Кожен PeerNode містить власне сховище sharedHtmlStorage, де зберігає оброблені веб-сторінки. Це усуває необхідність в центральній базі даних та зменшує навантаження на мережу.
2. Колективну обробку веб-контенту: Вузли можуть обмінюватися вже завантаженими сторінками через методи sendHtml() та receiveHtml(), уникаючи дублювання роботи та оптимізуючи використання ресурсів.
3. Автономність та масштабованість: Новий peer може приєднатися до мережі без необхідності змін в існуючих вузлах. Мережа автоматично масштабується з додаванням нових учасників.
4. Відмовостійкість: При виході з ладу одного вузла інші можуть продовжувати роботу, а дані зберігаються розподілено серед усіх учасників мережі.

У нашому випадку P2P архітектура забезпечує розподілене виконання завдань краулінгу, дозволяючи розподіляти навантаження по завантаженню сторінок між різними вузлами, реалізувати механізм обміну даними між краулерами та забезпечити високу доступність системи навіть при збоях окремих компонентів.

Структура архітектури

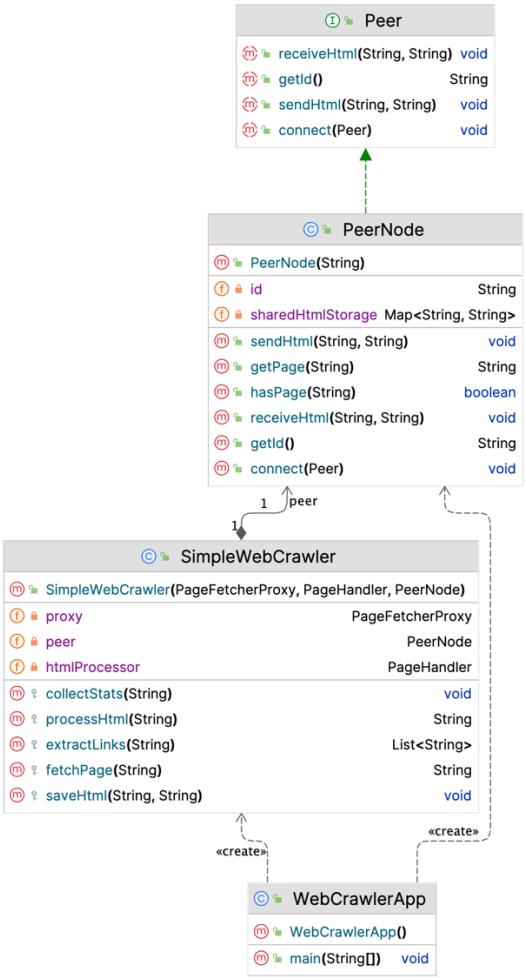

Рис. 5 – Структура архітектури Peer-to-Peer

Опис реалізації:

- Peer – інтерфейс, що визначає базову функціональність P2P вузла
- PeerNode – конкретна реалізація вузла з локальним сховищем HTML-сторінок
- SimpleWebCrawler – інтегрований з P2P архітектурою через параметр PeerNode
- connect() – метод для встановлення зв'язків між вузлами
- sendHtml()/receiveHtml() – методи для обміну обробленими сторінками

Такий підхід особливо ефективний для веб–краулерів, де необхідно обробляти великі обсяги даних з різних джерел, важлива висока доступність та відмовостійкість системи, потрібно розподіляти навантаження між багатьма екземплярами та за можливості уникати блокування через часті запити до одних і тих самих сайтів.

Посилання на репозиторій: https://github.com/dariahorlach/Lab–TRPZ

Висновки

Висновки: під час виконання лабораторної роботи, було реалізовано Peer-to-Peer архітектуру для веб-краулера, що дозволило створити розподілену децентралізовану систему збору веб-даних. Клас PeerNode реалізував автономний вузол мережі, здатний одночасно виконувати функції краулера, обробника контенту та розподіленого сховища даних. Реалізація продемонструвала, що P2P архітектура ефективно вирішує завдання масштабування веб-краулінгу шляхом усування єдиної точки відмови через децентралізовану структуру мережі та розподілу навантаження чи обробці сторінок між різними вузлами. Архітектура з P2P інтегрувалася з усією існуючою системою веб-краулера, де SimpleWebCrawler тепер приймає параметр PeerNode для зберігання результатів у розподіленому сховищі. Це дозволило створити гібридну систему, що поєднує переваги патернів проектування з перевагами розподіленої архітектури.

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 – 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.