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"

Виконав

студент групи ІА–33:

Китченко Д.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Вступ

Сучасні програмні системи часто потребують роботи в розподіленому середовищі, де різні компоненти взаємодіють через мережу. Існує три основні архітектурні підходи до організації такої взаємодії: клієнт-серверна архітектура, однорангові мережі (Peer-to-Peer) та сервіс-орієнтована архітектура.

У даній лабораторній роботі реалізовано P2P архітектуру для системи архівації файлів. P2P підхід обрано тому, що він найкраще відповідає задачі обміну архівами між користувачами. На відміну від клієнт-серверної моделі, де центральний сервер є точкою відмови та вузьким місцем, P2P дозволяє кожному учаснику бути одночасно і постачальником і споживачем ресурсів.

Ключова особливість реалізованої системи полягає в тому, що кожен вузол мережі містить як серверний компонент для обробки вхідних запитів, так і клієнтський компонент для підключення до інших вузлів. Це забезпечує децентралізовану архітектуру, де вихід з ладу одного вузла не впливає на роботу решти мережі.

Практична цінність P2P архітектури для архіватора полягає у можливості прямого обміну архівами між користувачами без необхідності завантажувати файли на центральний сервер. Це економить трафік, підвищує швидкість передачі та забезпечує кращу масштабованість системи.

Мета роботи

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

Конкретні завдання:

- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії
- Реалізувати функціонал для роботи в розподіленому оточенні
- Реалізувати взаємодію розподілених частин для P2P архітектури
- Підготувати діаграму класів спроєктованої архітектури

Теоретичні відомості

Peer-to-Peer архітектура

P2P архітектура - це модель мережевої взаємодії, в якій кожен вузол є одночасно клієнтом і сервером. Всі вузли мають рівні права та можливості для обміну даними. На відміну від клієнт-серверної моделі, де є чітке розділення ролей, P2P дозволяє учасникам взаємодіяти безпосередньо без централізованого сервера.

Децентралізація означає відсутність центрального сервера. Це зменшує залежність від одного вузла та підвищує стійкість мережі до збоїв. Якщо один комп'ютер вимкнеться, решта продовжують працювати.

Рівноправність вузлів означає що кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси). Немає "головного" комп'ютера.

Розподіл ресурсів передбачає що вузли надають доступ до своїх власних ресурсів: обчислювальна потужність, дисковий простір або файли.

Приклади використання P2P

BitTorrent використовує P2P для файлообміну. При скачуванні файл отримується шматочками від багатьох користувачів одночасно. Криптовалюти типу Bitcoin працюють на P2P принципі, де всі учасники зберігають копію блокчейну. Skype раніше використовував P2P для голосових дзвінків, передаючи голос напряму між користувачами.

Порівняння з клієнт-серверною архітектурою

У клієнт-серверній архітектурі є чітке розділення: сервер зберігає та обробляє дані, клієнти лише відображають інформацію. Переваги: простота управління, централізований контроль. Недоліки: сервер є точкою відмови, навантаження на один вузол.

У P2P архітектурі навантаження розподілене між всіма учасниками. Переваги: стійкість до збоїв, масштабованість. Недоліки: складність синхронізації, проблеми безпеки.

Технології реалізації P2P в Java

Для реалізації P2P комунікації в Java використовуються:

- java.net.Socket - встановлення TCP з'єднання між вузлами
- java.net.ServerSocket - прийом вхідних підключень
- ObjectInputStream/ObjectOutputStream - серіалізація об'єктів для передачі через мережу
- java.lang.Thread - багатопоточність для одночасної обробки кількох клієнтів

Хід роботи

Моя тема

14. **Архіватор** (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Архітектура системи

Для реалізації P2P архіватора створено 6 класів:

PeerNode - головний клас що представляє вузол P2P мережі. Містить всередині і сервер і клієнт, що дозволяє вузлу одночасно роздавати та отримувати архіви.

PeerServer - серверний компонент що приймає підключення від інших вузлів. Працює в окремому потоці та обробляє запити на отримання списку архівів та завантаження файлів.

PeerClient - клієнтський компонент для підключення до інших вузлів. Відправляє запити та отримує відповіді.

ArchiveInfo - модель даних про архів (назва, розмір, формат). Реалізує Serializable для передачі через мережу.

MessageType - перелік типів повідомлень між вузлами.

PeerDemo - демонстраційний клас що показує роботу системи.

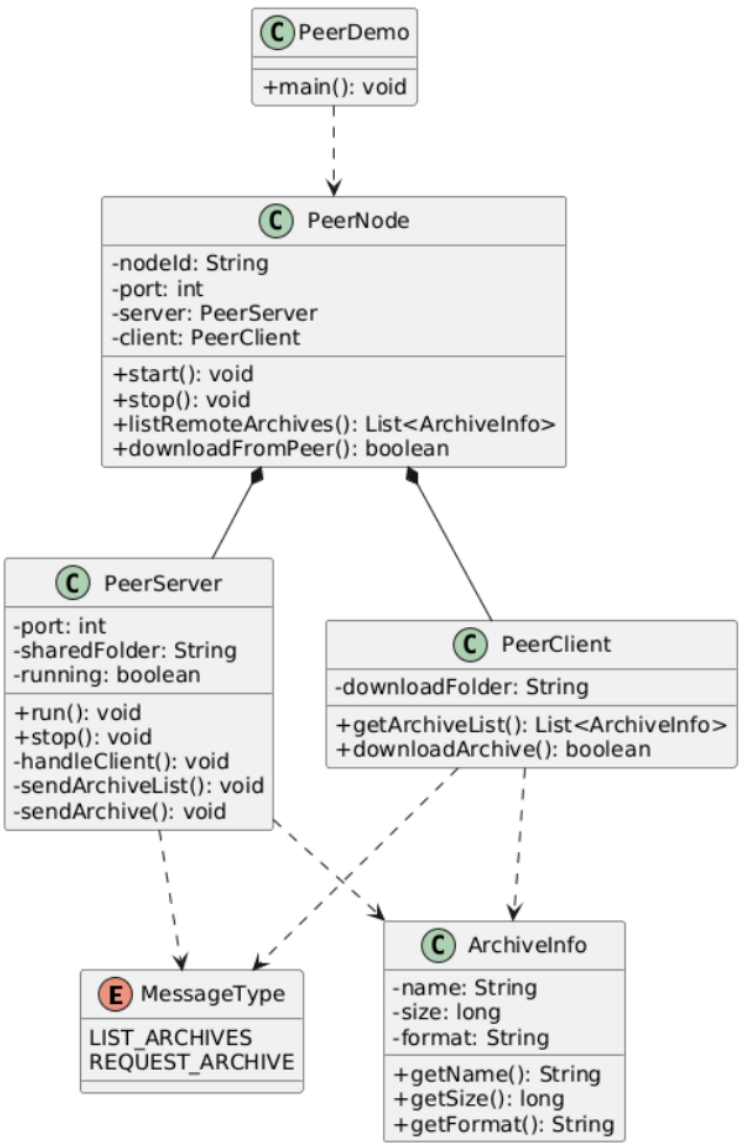


Рисунок 1. Діаграма класів

Реалізація ключових компонентів

Клас ArchiveInfo - модель даних про архів:

```
package p2p;

import java.io.Serializable;

public class ArchiveInfo implements Serializable {
    private String name;
    private long size;
    private String format;

    public ArchiveInfo(String name, long size, String format) {
        this.name = name;
        this.size = size;
        this.format = format;
    }

    public String getName() {
        return name;
    }
```

```
    public long getSize() {
        return size;
    }

    public String getFormat() {
        return format;
    }

    @Override
    public String toString() {
        return name + " (" + size + " bytes, " + format + ")";
    }
}
```

Клас реалізує Serializable, що дозволяє Java автоматично перетворювати об'єкт в байти для передачі через мережу та відновлювати його на іншому кінці.

Поля класу:

- name - назва файлу архіву
- size - розмір в байтах
- format - тип архіву (ZIP або TAR.GZ)

Метод toString() повертає зручне для читання представлення інформації про архів.

Клас MessageType - типи повідомлень:

```
package p2p;

public enum MessageType {
    LIST_ARCHIVES,  // запит "дай мені список архівів"
    REQUEST_ARCHIVE   // запит на отримання архіву
}
```

Використання enum забезпечує типобезпечність. Сервер перевіряє тип повідомлення та виконує відповідну дію.

Клас PeerClient - підключення до інших вузлів:

```
package p2p;

import java.io.*;
import java.net.*;
import java.util.*;

public class PeerClient {

    private String downloadFolder;  // куди зберігати скачані архіви

    public PeerClient(String downloadFolder) {
        this.downloadFolder = downloadFolder;
        new File(downloadFolder).mkdirs();
    }


    public List<ArchiveInfo> getArchiveList(String host, int port) {
        try (Socket socket = new Socket(host, port);
             ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
```

```
             ObjectInputStream in = new
ObjectInputStream(socket.getInputStream())) {

            // відправляємо запит
            out.writeObject(MessageType.LIST_ARCHIVES);
            out.flush();

            // отримуємо відповідь
            List<ArchiveInfo> archives = (List<ArchiveInfo>) in.readObject();
            System.out.println("Отримано список архівів від " + host + ":" +
port);
            return archives;

        } catch (Exception e) {
            System.err.println("Помилка підключення до " + host + ":" + port);
            e.printStackTrace();
            return new ArrayList<>();
        }
    }


    public boolean downloadArchive(String host, int port, String archiveName) {
        try (Socket socket = new Socket(host, port);
             ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new
ObjectInputStream(socket.getInputStream())) {

            // відправляємо запит на конкретний архів
            out.writeObject(MessageType.REQUEST_ARCHIVE);
            out.writeObject(archiveName);
            out.flush();

            // отримуємо відповідь
            String status = (String) in.readObject();

            if (status.equals("OK")) {
                byte[] fileData = (byte[]) in.readObject();

                // зберігаємо файл
                File outputFile = new File(downloadFolder, archiveName);
                try (FileOutputStream fos = new FileOutputStream(outputFile)) {
                    fos.write(fileData);
                }

                System.out.println("Скачано архів: " + archiveName + " (" +
fileData.length + " bytes)");
                return true;
            } else {
                System.err.println("Помилка: " + status);
                return false;
            }

        } catch (Exception e) {
            System.err.println("Помилка скачування: " + e.getMessage());
            return false;
        }
    }
}
```

Socket створює TCP з'єднання з іншим комп'ютером. ObjectOutputStream перетворює Java об'єкти в байти (серіалізація). ObjectInputStream відновлює

об'єкти з байтів (десеріалізація). Try-with-resources автоматично закриває ресурси.

Клас PeerServer - прийом підключень:

```
package p2p;

import java.io.*;
import java.net.*;
import java.util.*;

public class PeerServer implements Runnable {

    private int port;
    private String sharedFolder; // папка з архівами які ділимо
    private boolean running = true;

    public PeerServer(int port, String sharedFolder) {
        this.port = port;
        this.sharedFolder = sharedFolder;
    }

    @Override
    public void run() {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("P2P Server запущено на порту " + port);

            while (running) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Підключився: " +
clientSocket.getInetAddress());

                // обробка кожного клієнта в окремому потоці
                new Thread(() -> handleClient(clientSocket)).start();
            }
        } catch (IOException e) {
            System.err.println("Помилка сервера: " + e.getMessage());
        }
    }

    private void handleClient(Socket socket) {
        try (ObjectInputStream in = new
ObjectInputStream(socket.getInputStream());
             ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream())) {

            // тип запиту
            MessageType messageType = (MessageType) in.readObject();

            switch (messageType) {
                case LIST_ARCHIVES:
                    // відправляємо список наших архівів
                    sendArchiveList(out);
                    break;

                case REQUEST_ARCHIVE:
                    // відправляємо конкретний архів
                    String archiveName = (String) in.readObject();
                    sendArchive(out, archiveName);
                    break;

                default:
                    System.out.println("Невідомий тип повідомлення");
```

```
            }

        } catch (Exception e) {
            System.err.println("Помилка обробки клієнта: " + e.getMessage());
        }
    }

    private void sendArchiveList(ObjectOutputStream out) throws IOException {
        List<ArchiveInfo> archives = new ArrayList<>();

        File folder = new File(sharedFolder);
        if (folder.exists() && folder.isDirectory()) {
            for (File file : folder.listFiles()) {
                if (file.getName().endsWith(".zip") ||
file.getName().endsWith(".tar.gz")) {
                    String format = file.getName().endsWith(".zip") ? "ZIP" :
"TAR.GZ";
                    archives.add(new ArchiveInfo(file.getName(), file.length(),
format));
                }
            }
        }

        out.writeObject(archives);
        out.flush();
        System.out.println("Відправлено список з " + archives.size() + "
архівів");
    }

    private void sendArchive(ObjectOutputStream out, String archiveName) throws
IOException {
        File archiveFile = new File(sharedFolder, archiveName);

        if (!archiveFile.exists()) {
            out.writeObject("ERROR: Архів не знайдено");
            return;
        }

        // читаємо файл в байти
        byte[] fileData = new byte[(int) archiveFile.length()];
        try (FileInputStream fis = new FileInputStream(archiveFile)) {
            fis.read(fileData);
        }

        out.writeObject("OK");
        out.writeObject(fileData);
        out.flush();
        System.out.println("Відправлено архів: " + archiveName);
    }

    public void stop() {
        running = false;
    }
}
```

ServerSocket слухає на порту та чекає підключень. Метод accept() блокує виконання поки хтось не підключиться. Кожен клієнт обробляється в окремому Thread, що дозволяє серверу одночасно обслуговувати багатьох клієнтів.

Клас PeerNode - об'єднання клієнта і сервера:

```
package p2p;
```

```
import java.util.*;


public class PeerNode {

    private String nodeId;
    private int port;
    private String sharedFolder;
    private String downloadFolder;

    private PeerServer server;
    private PeerClient client;
    private Thread serverThread;

    public PeerNode(String nodeId, int port, String sharedFolder, String
downloadFolder) {
        this.nodeId = nodeId;
        this.port = port;
        this.sharedFolder = sharedFolder;
        this.downloadFolder = downloadFolder;

        this.server = new PeerServer(port, sharedFolder);
        this.client = new PeerClient(downloadFolder);
    }


    public void start() {
        serverThread = new Thread(server);
        serverThread.start();
        System.out.println("Вузол " + nodeId + " запущено на порту " + port);
    }


    public void stop() {
        server.stop();
        System.out.println("Вузол " + nodeId + " зупинено");
    }


    public List<ArchiveInfo> listRemoteArchives(String host, int port) {
        System.out.println("Запитую список архівів у " + host + ":" + port);
        return client.getArchiveList(host, port);
    }


    public boolean downloadFromPeer(String host, int port, String archiveName) {
        System.out.println("Скачую " + archiveName + " з " + host + ":" + port);
        return client.downloadArchive(host, port, archiveName);
    }

    public String getNodeId() {
        return nodeId;
    }

    public int getPort() {
        return port;
    }
}
```

PeerNode об'єднує PeerServer та PeerClient в одному об'єкті. Це реалізує головний принцип P2P - кожен вузол одночасно і роздає ресурси (через сервер)

і споживає ресурси (через клієнта). Сервер запускається в окремому потоці щоб постійно чекати підключень.

Метод start() створює новий Thread для сервера і запускає його. Це важливо бо сервер працює в безкінечному циклі і заблокував би основну програму.

Методи listRemoteArchives() та downloadFromPeer() є обгортками над методами клієнта. Вони спрощують використання вузла, приховуючи внутрішню структуру.

Клас PeerDemo - демонстрація:

```
package p2p;

import java.util.*;

public class PeerDemo {

    public static void main(String[] args) throws Exception {

        System.out.println("P2P АРХІВАТОР\n");

        // створюємо два вузли (як два комп'ютери)
        // вузол 1 має архіви в папці shared1, скачує в downloads1
        PeerNode node1 = new PeerNode("Node-1", 8081, "shared1", "downloads1");

        // вузол 2 має архіви в папці shared2, скачує в downloads2
        PeerNode node2 = new PeerNode("Node-2", 8082, "shared2", "downloads2");

        // запускаємо обидва вузли
        node1.start();
        node2.start();

        // трохи затримки щоб сервери запустились
        Thread.sleep(1000);

        System.out.println("\n Node 2 запитує список архівів у Node 1");
        List<ArchiveInfo> node1Archives = node2.listRemoteArchives("localhost",
8081);

        System.out.println("Архіви на Node 1:");
        for (ArchiveInfo archive : node1Archives) {
            System.out.println(" - " + archive);
        }

        System.out.println("\n Node 1 запитує список архівів у Node 2");
        List<ArchiveInfo> node2Archives = node1.listRemoteArchives("localhost",
8082);

        System.out.println("Архіви на Node 2:");
        for (ArchiveInfo archive : node2Archives) {
            System.out.println(" - " + archive);
        }

        // скачуємо архів якщо є
        if (!node1Archives.isEmpty()) {
            String archiveName = node1Archives.get(0).getName();
            System.out.println("\n Node 2 скачує '" + archiveName + "' з Node
1");
```

```
            boolean success = node2.downloadFromPeer("localhost", 8081,
archiveName);
            if (success) {
                System.out.println("Результат: " + "УСПІХ");
            } else {
                System.out.println("Результат: " + "ПОМИЛКА");
            }
        }
        if (!node2Archives.isEmpty()) {
            String archiveName = node2Archives.get(0).getName();
            System.out.println("\n Node 1 скачує '" + archiveName + "' з Node
2");
            boolean success = node1.downloadFromPeer("localhost", 8082,
archiveName);
            if (success) {
                System.out.println("Результат: " + "УСПІХ");
            } else {
                System.out.println("Результат: " + "ПОМИЛКА");
            }
        }

        // зупиняємо вузли
        Thread.sleep(2000);
        node1.stop();
        node2.stop();

        System.out.println("\n КІНЕЦЬ");
    }
}
```

Результати роботи

P2P АРХІВАТОР

Вузол Node-1 запущено на порту 8081

Вузол Node-2 запущено на порту 8082

P2P Server запущено на порту 8081

P2P Server запущено на порту 8082

Node 2 запитує список архівів у Node 1

Запитую список архівів у localhost:8081

Підключився: /127.0.0.1

Отримано список архівів від localhost:8081

Відправлено список з 1 архівів

Архіви на Node 1:

- photos.zip (71706 bytes, ZIP)

Node 1 запитує список архівів у Node 2

Запитую список архівів у localhost:8082

Підключився: /127.0.0.1

Відправлено список з 1 архівів

Отримано список архівів від localhost:8082

Архіви на Node 2:

- test.zip (974 bytes, ZIP)

Node 2 скачує 'photos.zip' з Node 1

Скачую photos.zip з localhost:8081

Підключився: /127.0.0.1

Відправлено архів: photos.zip

Скачано архів: photos.zip (71706 bytes)

Результат: УСПІХ

Node 1 скачує 'test.zip' з Node 2

Скачую test.zip з localhost:8082

Підключився: /127.0.0.1

Відправлено архів: test.zip

Скачано архів: test.zip (974 bytes)

Результат: УСПІХ

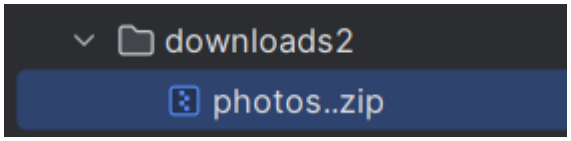
Вузол Node-1 зупинено

Вузол Node-2 зупинено

КІНЕЦЬ

Адреса 127.0.0.1 (localhost) означає що обидва вузли працюють на одному комп'ютері для тестування. В реальному сценарії вузли працювали б на різних комп'ютерах з різними IP адресами.

Після виконання програми файл photos.zip з'явився в папці downloads2, що підтверджує успішну передачу файлу між вузлами.



Висновки

В ході виконання даної лабораторної роботи було реалізовано Peer-to-Peer архітектуру для системи архівації файлів. Створено 6 класів що забезпечують повноцінну P2P взаємодію: PeerNode як головний клас вузла, PeerServer для прийому підключень, PeerClient для відправки запитів, ArchiveInfo для представлення даних про архів, MessageType для типізації повідомлень та PeerDemo для демонстрації.

Ключові технічні рішення:

Використано TCP сокети (java.net.Socket та ServerSocket) для надійної передачі даних між вузлами. TCP гарантує доставку пакетів у правильному порядку.

Застосовано серіалізацію об'єктів (ObjectInputStream та ObjectOutputStream) що дозволяє передавати складні структури даних через мережу без ручного перетворення.

Реалізовано багатопоточність через Thread для обробки кожного клієнта. Це дозволяє серверу обслуговувати декількох клієнтів одночасно не блокуючи один одного.

Архітектура відповідає принципам P2P: децентралізація (немає центрального сервера), рівноправність вузлів (кожен і роздає і качає) та розподіл ресурсів (кожен вузол ділиться своїми архівами).

Система успішно демонструє обмін архівами між вузлами, отримання списків доступних файлів та завантаження обраних архівів.

Питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?

Модель де є два типи додатків: сервер (зберігає та обробляє дані) і клієнти (відображають інформацію користувачу). Клієнт відправляє запит -> сервер обробляє -> повертає відповідь. Приклад: веб-браузер (клієнт) і веб-сервер.

2. Розкажіть про сервіс-орієнтовану архітектуру.

SOA - підхід де система складається з незалежних сервісів зі стандартизованими інтерфейсами. Кожен сервіс виконує конкретну бізнес-функцію. Сервіси взаємодіють через обмін повідомленнями (зазвичай HTTP/SOAP/REST). Альтернатива монолітній архітектурі.

3. Якими принципами керується SOA?

- Слабка зв'язність (loose coupling) - сервіси незалежні один від одного
- Стандартизовані інтерфейси - єдиний спосіб комунікації
- Повторне використання - сервіс можна використовувати в різних застосунках
- Автономність - кожен сервіс самодостатній
- Абстракція - приховування внутрішньої реалізації

4. Як між собою взаємодіють сервіси в SOA?

Через обмін повідомленнями по стандартизованих протоколах. Зазвичай HTTP з використанням SOAP або REST. Часто через централізовану шину даних (Enterprise Service Bus). Сервіси не мають прямого доступу до баз даних інших сервісів.

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

Сервіси реєструються в спеціальних реєстрах (Service Registry). Розробники шукають потрібний сервіс в реєстрі, отримують його опис (WSDL для SOAP або OpenAPI для REST) з інформацією про ендпоінти, формат запитів та відповідей.

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

Переваги:

- Централізоване управління даними
- Простота оновлення (оновлюється тільки сервер)
- Легкий контроль безпеки
- Простіша підтримка

Недоліки:

- Сервер - точка відмови (впав сервер = все не працює)
- Високе навантаження на сервер
- Потрібен потужний сервер
- Проблеми масштабування

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

Переваги:

- Децентралізація (немає точки відмови)
- Розподіл навантаження між вузлами
- Масштабованість (більше вузлів = більше ресурсів)
- Стійкість до збоїв

Недоліки:

- Складність синхронізації даних
- Проблеми з безпекою (важко контролювати)
- Складний пошук ресурсів в мережі

Важко гарантувати доступність ресурсу

8. Що таке мікро-сервісна архітектура?

Підхід до створення серверного додатку як набору малих незалежних служб. Кожен мікросервіс:

- Реалізує одну бізнес-функцію
- Має власну базу даних
- Розгортається незалежно
- Взаємодіє з іншими через API (HTTP/HTTPS, WebSockets, AMQP)

Розвиток SOA з більш дрібними та автономними сервісами.

9. Які протоколи використовуються для обміну даними в мікросервісній

архітектурі?

- HTTP/HTTPS - найпоширеніший
- REST API - архітектурний стиль поверх HTTP
- gRPC - швидкий бінарний протокол від Google
- WebSockets - двосторонній зв'язок в реальному часі
- AMQP (RabbitMQ) - черги повідомлень
- Apache Kafka - потокова передача даних

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 багатошарова архітектура (layered architecture), а не SOA.

SOA передбачає:

- Розподілені сервіси (окремі процеси/сервери)
- Мережеву взаємодію між сервісами
- Незалежне розгортання

Шар сервісів всередині одного додатку - це просто розділення відповідальностей, а не сервіс-орієнтована архітектура. Всі "сервіси" працюють в одному процесі і викликаються напряму, а не через мережу.