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

із дисципліни «**Технології розроблення програмного забезпечення**»

“РІЗНІ ВИДИ ВЗАЄМОДІЇ ДОДАТКІВ: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE”

Варіант №5. Аудіоредактор

Виконала
студентка групи ІА–31
Биковська О.І.

Перевірив
викладач
Мягкий М. Ю.

Київ 2025

Зміст

Мета: Вивчення різних видів взаємодії додатків: client-server, peer-to-peer, service-oriented architecture при створенні проєкту за індивідуальним варіантом.

Тема (Варіант №5)

5 Аудіо редактор (singleton, adapter, observer, mediator, composite, client-server) Аудіо редактор повинен володіти наступним функціоналом: представлення аудіо даних будь-якого формату в WAVE-формі, вибір і подальші операції копіювання / вставки / вирізання / деформації по сегменту аудіозапису, можливість роботи з декількома звуковими доріжками, кодування в найбільш поширених форматах (ogg, flac, mp3).

Хід роботи

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Реалізувати взаємодію програми в одній з архітектур відповідно до обраної теми.

Теоретичні відомості

Клієнт-серверні додатки поділяються на тонкі та товсті клієнти залежно від розподілу обчислювальної роботи між клієнтською та серверною частинами. У тонких клієнтах основна логіка обробки виконується на сервері, що зменшує вимоги до клієнтських пристроїв, але збільшує навантаження на сервер. Такий підхід зручний для сценаріїв із підвищеними вимогами до безпеки або при роботі з великим обсягом даних і кількістю користувачів. У товстих клієнтах більшість обчислень виконується на клієнтській стороні, що зменшує навантаження на сервер, але вимагає більш потужного клієнтського обладнання.

Однією з поширених моделей у клієнт-серверній архітектурі є "підписка/видача", яка забезпечує автоматичне отримання оновлень без постійного запиту до сервера. Ця модель зазвичай реалізується через трирівневу структуру, що включає клієнтську частину для роботи з інтерфейсом, проміжну частину (middleware) для спільних компонентів і серверну частину для обробки бізнес-логіки, зберігання даних і їх передачі.

У Peer-to-Peer (P2P) додатках усі клієнти мають рівноправний статус, а сервер відсутній. Основними викликами в P2P-додатках є синхронізація даних між клієнтами та організація пошуку доступних вузлів. Для вирішення цих питань застосовують централізовані адреси для пошуку або алгоритми прямої синхронізації, наприклад, із використанням хешування. P2P-додатки зазвичай використовують спеціальні протоколи та формати даних.

Сервіс-орієнтована архітектура (SOA) забезпечує побудову додатків із незалежних компонентів, які взаємодіють через стандартизовані інтерфейси, такі як SOAP або REST. Цей підхід дозволяє створювати масштабовані, платформонезалежні системи, які легко інтегруються та повторно використовуються.

Програмне забезпечення як послуга (SaaS) – це модель надання програмного забезпечення через Інтернет, яка дозволяє користувачам отримувати доступ до додатків без необхідності їх інсталяції. SaaS забезпечує централізоване

оновлення, технічну підтримку та оплату за підпискою або обсягом використання. Реалізація SaaS зазвичай базується на сервіс-орієнтованій архітектурі та хмарних технологіях із використанням stateless-сервісів.

Мікросервісна архітектура є підходом до побудови серверних додатків у вигляді набору незалежних служб, кожна з яких виконує конкретну бізнес-функцію. Ці служби працюють у своїх процесах і взаємодіють через стандартизовані протоколи, такі як HTTP, WebSockets або AMQP. Такий підхід забезпечує масштабованість, гнучкість і незалежність розробки та оновлення окремих компонентів системи, що робить його ефективним для створення складних корпоративних додатків.

Клієнт-серверна взаємодія

Структура та обгрунтування вибору.

Ознайомившись з теоретичними матеріалами, було прийнято рішення реалізувати взаємодію між клієнтом і сервером за принципом клієнт-серверної архітектури. У цій архітектурі сервер відповідає за виконання всіх основних обчислювальних операцій, зокрема, обробку файлів, та повертає результати клієнту. Клієнт, у свою чергу, виступає як інтерфейс користувача, через який відправляються файли на сервер для обробки та отримуються результати. Взаємодія між клієнтом і сервером відбуватиметься через мережу за допомогою інтерфейсу input/output (I/O), наприклад, використовуючи протокол HTTP. Загальну структуру графічно описано на Рисунку 1.

Рисунок 1. Загальний вигляд «спілкування» у клієнт-серверної архітектури

Для початку комунікації необхідно запустити сервер

```
ServerSocket serverSocket = new ServerSocket( port: 55555);
System.out.println("Server started on port 55555");

Socket clientSocket = serverSocket.accept();
System.out.println("Client connected: " + clientSocket.getInetAddress());


ObjectInputStream objectInputStream = new ObjectInputStream(clientSocket.getInputStream());
ObjectOutputStream objectOutputStream = new ObjectOutputStream(clientSocket.getOutputStream());
```

Рисунок 2. Сервер запускається на порті номер 55555 та починає «слухати» до підключення клієнта

Після підключення клієнта сервер починає чекати на команди клієнта. Після отримання такої команди, сервер реагує на неї відповідним чином (наприклад, обчислює wave-форму аудіо) та повертає дані клієнту

Приклад комунікації між клієнтом та сервером з використанням команди «вирізати»:

```
cutButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        cut();
    }
});
```

Рисунок 3. До кнопки «cut» підключено «слухач», який виконує певний код, кожен раз коли кнопка натискається

Клієнт надсилає серверу команду «вирізати» (код методу sendCutCommand() див. нижче) та значення крайніх точок повзунка. Після обробки та обчислення інформації (див. нижче) сервер повертає значення амплітуди, який клієнтська сторона приймає через метод інтерфейсу input/output objectInputStream.readObject(), і далі працює з цим значенням самостійно, відображаючи його у графічному інтерфейсі користувача.

```
private void cut() {
    try {
        sendCutCommand(rangeSlider.getValue(), rangeSlider.getUpperValue());

        int[] data = (int[]) objectInputStream.readObject();
        wavePanel.setData(data);
        SwingUtilities.updateComponentTreeUI(contentPane);

        rangeSlider.setMinimum(0);
        rangeSlider.setMaximum(data.length);

        rangeSlider.setValue(((int) (rangeSlider.getMaximum() * 0.5)));
        rangeSlider.setUpperValue(((int) (rangeSlider.getMaximum() * 0.75)));
    } catch (IOException | ClassNotFoundException e) {
        throw new RuntimeException(e);
    }
}
```

Рисунок 4. Код методу cut()

```
private void sendCutCommand(int l, int u) throws IOException {
    objectOutputStream.writeObject("cut");
    objectOutputStream.writeObject(l);
    objectOutputStream.writeObject(u);
}
```

Рисунок 5. Метод sendCutCommand(), який відправляє на сервер команду для виконання та значення повзунка

Спочатку приймається два значення, відправлені клієнтом. Ці значення передаються методу сервера та результат виконання повертається клієнту:

```
} if (Objects.equals(command, b: "cut")) {
    int l = (int) objectInputStream.readObject();
    int u = (int) objectInputStream.readObject();
    cutAudio(l, u);
    objectOutputStream.writeObject(waveData.extractAmplitudeFromFile(file));
```

Рисунок 6. Код, який виконує сервер, коли користувач надсилає команду «вирізати»

Відповідним чином реалізовано інші команди застосунку. Клієнт не бере участі у будь-якому обчисленні даних, а лише у їх відправці, прийманні та візуалізації

Реалізація функціоналу у коді з використанням архітектури

Server

```
package org.example.server;

import org.example.audiotrack.Wav;
import org.example.audiotrack.WaveData;
import org.example.converter.AudioFlacConverter;
import org.example.converter.AudioMp3Converter;
import org.example.converter.AudioOggConverter;

import javax.sound.sampled.AudioFileFormat;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Objects;

public class Server {

    private static File file;
    private static File temp;

    public static void main(String[] args) {

        try {


            ServerSocket serverSocket = new ServerSocket(55555);
            System.out.println("Server started on port 55555");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " +
clientSocket.getInetAddress());


            ObjectInputStream objectInputStream = new
ObjectInputStream(clientSocket.getInputStream());
            ObjectOutputStream objectOutputStream = new
ObjectOutputStream(clientSocket.getOutputStream());

            WaveData waveData = new WaveData();




            while (true) {
                String command = (String) objectInputStream.readObject();

                if (Objects.equals(command, "select")) {
                    file = (File) objectInputStream.readObject();

objectOutputStream.writeObject(waveData.extractAmplitudeFromFile(file));
                } else if (Objects.equals(command, "copy")) {
```

```
                    int l = (int) objectInputStream.readObject();
                    int u = (int) objectInputStream.readObject();
                    copyAudio(l, u);

objectOutputStream.writeObject(waveData.extractAmplitudeFromFile(file));
                } else if (Objects.equals(command, "cut")) {
                    int l = (int) objectInputStream.readObject();
                    int u = (int) objectInputStream.readObject();
                    cutAudio(l, u);

objectOutputStream.writeObject(waveData.extractAmplitudeFromFile(file));
                } else if (Objects.equals(command, "paste")) {
                    double x = (double) objectInputStream.readObject();
                    pasteAudio(x);

objectOutputStream.writeObject(waveData.extractAmplitudeFromFile(file));
                } else if (Objects.equals(command, "convertToMp3")) {
                    AudioMp3Converter converter =
AudioMp3Converter.getInstance();
                    Wav wav = new Wav(file.getPath());
                    File mp3 = converter.convertTo(wav);
                    mp3.createNewFile();
                } else if (Objects.equals(command, "convertToOgg")) {
                    AudioOggConverter converter =
AudioOggConverter.getInstance();
                    Wav wav = new Wav(file.getPath());
                    File ogg = converter.convertTo(wav);
                    ogg.createNewFile();
                } else if (Objects.equals(command, "convertToFlac")) {
                    AudioFlacConverter converter =
AudioFlacConverter.getInstance();
                    Wav wav = new Wav(file.getPath());
                    File flac = converter.convertTo(wav);
                    flac.createNewFile();
                }
            }

        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }


    public static void copyAudio(int startByte, int endByte) {
        AudioInputStream inputStream = null;
        AudioInputStream shortenedStream = null;
        try {
            AudioFileFormat fileFormat = AudioSystem.getAudioFileFormat(file);
            AudioFormat format = fileFormat.getFormat();
            inputStream = AudioSystem.getAudioInputStream(file);
            inputStream.skip(startByte * 2L);
            shortenedStream = new AudioInputStream(inputStream, format, (endByte
- startByte));


            temp = new File("temp.wav");
            temp.deleteOnExit();
            AudioSystem.write(shortenedStream, fileFormat.getType(), temp);
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            if (inputStream != null) try {
                inputStream.close();
            } catch (Exception e) {
                System.out.println(e);
```

```
            }
            if (shortenedStream != null) try {
                shortenedStream.close();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }

    public static void cutAudio(int startByte, int endByte) {
        AudioInputStream inputStream = null;
        AudioInputStream inputStream2 = null;

        AudioInputStream firstPart = null;
        AudioInputStream secondPart = null;
        AudioInputStream ais = null;
        try {
            AudioFileFormat fileFormat = AudioSystem.getAudioFileFormat(file);
            AudioFormat format = fileFormat.getFormat();
            inputStream = AudioSystem.getAudioInputStream(file);
            inputStream2 = AudioSystem.getAudioInputStream(file);

            firstPart = new AudioInputStream(inputStream,
fileFormat.getFormat(), startByte);

            inputStream2.skip((endByte - startByte) * 2L);

            secondPart = new AudioInputStream(inputStream2,
fileFormat.getFormat(), inputStream2.available() / 2);

            SequenceInputStream sequenceInputStream = new
SequenceInputStream(firstPart, secondPart);
            ais = new AudioInputStream(sequenceInputStream, format,
secondPart.getFrameLength());

            File temp = new File("temp.wav");
            AudioSystem.write(ais, fileFormat.getType(), temp);
            copyFileUsingStream(temp, file);
            temp.delete();


        } catch (Exception e) {
            System.out.println(e);
        } finally {
            if (inputStream != null) try {
                inputStream.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (firstPart != null) try {
                firstPart.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (secondPart != null) try {
                secondPart.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (inputStream2 != null) try {
                inputStream2.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (ais != null) try {
```

```
                ais.close();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }

    public static void pasteAudio(double x) {
        if (temp == null) {
            throw new RuntimeException("Фрагмент для копіювання відсутній.");
        }
        AudioInputStream inputStream = null;
        AudioInputStream inputStream2 = null;
        AudioInputStream inputStream3 = null;

        AudioInputStream firstPart = null;
        AudioInputStream secondPart = null;
        AudioInputStream middlePart = null;
        AudioInputStream ais = null;

        try {
            AudioFileFormat fileFormat = AudioSystem.getAudioFileFormat(file);
            AudioFormat format = fileFormat.getFormat();
            inputStream = AudioSystem.getAudioInputStream(file);
            inputStream2 = AudioSystem.getAudioInputStream(file);
            inputStream3 = AudioSystem.getAudioInputStream(temp);

            int size = inputStream.available() / 2;
            int startByte = (int) (size * x);

            firstPart = new AudioInputStream(inputStream,
fileFormat.getFormat(), startByte);

            inputStream2.skip(startByte * 2L);
            secondPart = new AudioInputStream(inputStream2,
fileFormat.getFormat(), inputStream2.available() / 2);


            middlePart = new AudioInputStream(inputStream3,
fileFormat.getFormat(), inputStream3.available() / 2);

            SequenceInputStream sequenceInputStream = new
SequenceInputStream(firstPart, middlePart);
            SequenceInputStream sequenceInputStream2 = new
SequenceInputStream(sequenceInputStream, secondPart);

            ais = new AudioInputStream(sequenceInputStream2, format,
firstPart.getFrameLength() + middlePart.getFrameLength() +
secondPart.getFrameLength());

            File temp = new File("temp.wav");
            AudioSystem.write(ais, fileFormat.getType(), temp);
            copyFileUsingStream(temp, file);
            temp.delete();
        } catch (Exception e) {
            System.out.println(e);
        } finally {
            if (inputStream != null) try {
                inputStream.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (firstPart != null) try {
                firstPart.close();
            } catch (Exception e) {
```

```
                System.out.println(e);
            }
            if (secondPart != null) try {
                secondPart.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (inputStream2 != null) try {
                inputStream2.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (inputStream3 != null) try {
                inputStream3.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (ais != null) try {
                secondPart.close();
            } catch (Exception e) {
                System.out.println(e);
            }
            if (middlePart != null) try {
                middlePart.close();
            } catch (Exception e) {
                System.out.println(e);
            }
        }
    }

    private static void copyFileUsingStream(File source, File dest) throws
IOException {
        InputStream is = null;
        OutputStream os = null;
        try {
            is = new FileInputStream(source);
            os = new FileOutputStream(dest);
            byte[] buffer = new byte[1024];
            int length;
            while ((length = is.read(buffer)) > 0) {
                os.write(buffer, 0, length);
            }
        } finally {
            if (is != null) {
                is.close();
            }
            if (os != null) {
                os.close();
            }
        }
    }
}
```

Client

```
package org.example.swing;

import org.example.audiotrack.WaveData;
import org.example.slider.RangeSlider;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class Test extends JDialog {
    {
        Socket socket = null;
        try {
            socket = new Socket("localhost", 55555);

            objectOutputStream = new
ObjectOutputStream(socket.getOutputStream());
            objectInputStream = new ObjectInputStream(socket.getInputStream());

        } catch (IOException e) {
            System.out.println("Server is not running.");
            System.exit(1);
        }
    }

    private JPanel contentPane;
    private JButton button;
    private JPanel wave;
    private WavePanel wavePanel;
    private JButton copyButton;
    private JButton convertToButton;
    private JButton pasteButton;
    private JButton cutButton;
    private RangeSlider rangeSlider;
    private ObjectOutputStream objectOutputStream;
    private ObjectInputStream objectInputStream;

    public Test() {
        setContentPane(contentPane);
        setModal(true);
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                select();
            }
        });
        copyButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                copy();
            }
        });
        cutButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                cut();
            }
        });
        pasteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                paste();
            }
        });
    }

    private void select() {
        JFileChooser fileChooser = new JFileChooser();
```

```
        int state = fileChooser.showOpenDialog(null);
        if (state == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            fileChooser.setVisible(false);
            try {
                sendSelectCommand(selectedFile);
                int[] data = (int[]) objectInputStream.readObject();
                wavePanel.setData(data);
                SwingUtilities.updateComponentTreeUI(contentPane);

                rangeSlider.setMinimum(0);
                rangeSlider.setMaximum(data.length);

                rangeSlider.setValue(((int) (rangeSlider.getMaximum() * 0.5)));
                rangeSlider.setUpperValue(((int) (rangeSlider.getMaximum() *
0.75)));
                rangeSlider.setVisible(true);
            } catch (IOException | ClassNotFoundException e) {
                System.out.println(e.getMessage());
            }
        }
    }

    private void copy() {
        try {
            sendCopyCommand(rangeSlider.getValue(),
rangeSlider.getUpperValue());

            int[] data = (int[]) objectInputStream.readObject();
            wavePanel.setData(data);
            SwingUtilities.updateComponentTreeUI(contentPane);

            rangeSlider.setMinimum(0);
            rangeSlider.setMaximum(data.length);

            rangeSlider.setVisible(true);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }

    private void cut() {
        try {
            sendCutCommand(rangeSlider.getValue(), rangeSlider.getUpperValue());

            int[] data = (int[]) objectInputStream.readObject();
            wavePanel.setData(data);
            SwingUtilities.updateComponentTreeUI(contentPane);

            rangeSlider.setMinimum(0);
            rangeSlider.setMaximum(data.length);

            rangeSlider.setValue(((int) (rangeSlider.getMaximum() * 0.5)));
            rangeSlider.setUpperValue(((int) (rangeSlider.getMaximum() *
0.75)));
        } catch (IOException | ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    private void paste() {
        try {
            double x = wave.getMousePosition().getX() / wave.getWidth();
            System.out.println(x);
            sendPasteCommand(x);
```

```
            int[] data = (int[]) objectInputStream.readObject();
            wavePanel.setData(data);
            SwingUtilities.updateComponentTreeUI(contentPane);

            rangeSlider.setMinimum(0);
            rangeSlider.setMaximum(data.length);

            rangeSlider.setValue(((int) (rangeSlider.getMaximum() * 0.5)));
            rangeSlider.setUpperValue(((int) (rangeSlider.getMaximum() *
0.75)));
        } catch (Exception e) {
            System.out.println("Мишка знаходиться у неправильному місці. Будь
ласка, оберіть місце на звуковій доріжці.");
        }
    }

    private void convertTo(String format) {
        try {
            sendConvertToCommand(format);
            JOptionPane.showMessageDialog(null, "Файл було успішно форматовано у
" + format);
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }


    private void sendCopyCommand(int l, int u) throws IOException {
        objectOutputStream.writeObject("copy");
        objectOutputStream.writeObject(l);
        objectOutputStream.writeObject(u);
    }

    private void sendCutCommand(int l, int u) throws IOException {
        objectOutputStream.writeObject("cut");
        objectOutputStream.writeObject(l);
        objectOutputStream.writeObject(u);
    }

    private void sendSelectCommand(File file) throws IOException {
        objectOutputStream.writeObject("select");
        objectOutputStream.writeObject(file);
    }

    private void sendPasteCommand(double x) throws IOException {
        objectOutputStream.writeObject("paste");
        objectOutputStream.writeObject(x);
    }

    private void sendConvertToCommand(String format) throws IOException {
        objectOutputStream.writeObject("convertTo" + format);
    }

    private void createUIComponents() {
        wave = new JPanel();
        wavePanel = new WavePanel();
        rangeSlider = new RangeSlider();
        rangeSlider.setVisible(false);

        convertToButton = new JButton();

        JPopupMenu popupMenu = new JPopupMenu();
```

```
        JMenuItem mp3 = new JMenuItem("mp3");
        JMenuItem ogg = new JMenuItem("ogg");
        JMenuItem flac = new JMenuItem("flac");

        mp3.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                convertTo("Mp3");
            }
        });

        ogg.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                convertTo("Ogg");
            }
        });

        flac.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                convertTo("Flac");
            }
        });

        popupMenu.add(mp3);
        popupMenu.add(ogg);
        popupMenu.add(flac);

        convertToButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {;
                popupMenu.show(convertToButton, 0, convertToButton.getHeight());
            }
        });


    }

    public static void main(String[] args) {
        Test dialog = new Test();
        dialog.pack();
        dialog.setVisible(true);
        System.exit(0);
    }

}
```

Висновок

У ході виконання лабораторної роботи було досліджено основні аспекти клієнт-серверної архітектури, Peer-to-Peer додатків, сервіс-орієнтованої архітектури (SOA), моделі SaaS та мікросервісного підходу. Було розглянуто особливості організації тонких і товстих клієнтів, які визначають розподіл обчислювального навантаження між серверною та клієнтською сторонами.

Також було проаналізовано модель "підписка/видача", яка дозволяє клієнтам отримувати оновлення в режимі реального часу без необхідності постійних запитів до сервера.

У рамках Peer-to-Peer архітектури було вивчено методи синхронізації даних між вузлами та пошуку клієнтів, що дозволяють створювати додатки без централізованого сервера. Особливу увагу приділено сервіс-орієнтованій архітектурі, яка забезпечує інтеграцію незалежних компонентів через стандартизовані інтерфейси, такі як SOAP або REST, що є основою для створення масштабованих корпоративних систем.

Модель SaaS продемонструвала переваги віддаленого доступу до програмного забезпечення через Інтернет, включаючи централізоване оновлення, технічну підтримку та зручність для користувачів. Мікросервісна архітектура, яка дозволяє розробляти серверні додатки як набір малих незалежних служб, виявилася ефективним рішенням для складних систем, що потребують масштабованості та гнучкості.

Під час лабораторної роботи було закріплено теоретичні знання через практичну реалізацію клієнт-серверного додатка з використанням одного з розглянутих підходів. Це дозволило глибше зрозуміти особливості проектування архітектури програмних систем, які забезпечують надійність, продуктивність та зручність використання.

1. **Що таке SOA?** SOA (Service-Oriented Architecture) — це архітектурний підхід до розробки програмного забезпечення, де додаток будується як набір незалежних, слабко пов'язаних сервісів, що взаємодіють між собою через мережу за допомогою стандартизованих протоколів.
2. **Якими принципами керується SOA?** Ключовими принципами SOA є слабка зв'язність компонентів, повторне використання сервісів, автономність, інкапсуляція (приховування внутрішньої логіки),

відсутність збереження стану (statelessness) та стандартні інтерфейси взаємодії.

3. **У чому полягають переваги та недоліки клієнт-серверної моделі?** Перевагою є централізоване управління даними та безпекою, що полегшує адміністрування; недоліком є вразливість системи через наявність єдиної точки відмови (сервера) та потенційні проблеми з продуктивністю при великому навантаженні.
4. **У чому полягають переваги та недоліки однорангової моделі взаємодії?** Перевагами є висока відмовостійкість і легка масштабованість, оскільки кожен вузол є рівноправним; недоліками є складність адміністрування, проблеми з безпекою даних та відсутність централізованого контролю.