Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №9
Технологія розробки програмного забезпечення
«Взаємодія компонентів системи»

Виконала
студентка групи ІА–32:
Ткачук М. С.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

Тема: Взаємодія компонентів системи

Мета: Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізувати в проєктованій системі одну із архітектур

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.

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).

Теоретичні відомості

Клієнт-серверна архітектура

Клієнт-серверна архітектура описує взаємодію між двома типами додатків: клієнтом, який відповідає за інтерфейс користувача, та сервером, який зберігає й опрацьовує дані. Виділяють тонких та товстих клієнтів. Тонкий клієнт передає основну логіку на сервер і містить лише інтерфейс, характерний для веб-застосунків. У таких системах більшість навантаження припадає саме на сервер, а оновлення спрощується, тому що достатньо оновити лише серверну частину. Товстий клієнт, навпаки, виконує основну логіку на своїй стороні, а сервер виступає лише як джерело даних. Такі клієнти працюють швидше та можуть частково функціонувати без підключення до сервера. Проміжним варіантом є SPA, де логіка виконується на клієнті, але додаток щоразу завантажується із сервера. Уся взаємодія клієнта і сервера зазвичай організована через трирівневу структуру: клієнтська частина, спільна (middleware) і серверна частина.

Peer-to-Peer архітектура

Peer-to-Peer архітектура базується на взаємодії рівноправних вузлів, де кожен пристрій може одночасно бути клієнтом і сервером. Відсутність централізації робить мережу стійкою до збоїв і дозволяє учасникам напряму обмінюватися даними та ресурсами. P2P широко застосовується у файлообмінниках, блокчейн-технологіях, VoIP та розподілених обчисленнях. Серед недоліків складність забезпечення безпеки, синхронізації та ефективного пошуку інформації в дуже великих мережах.

Сервіс-орієнтована архітектура

Сервіс-орієнтована архітектура це підхід до створення програмних систем, де функціональність розбита на незалежні сервіси зі стандартизованими інтерфейсами. Замість моноліту використовується набір розподілених служб, які взаємодіють через повідомлення, найчастіше через HTTP та протоколи SOAP або REST. Сервіси можуть бути як повноцінними модулями, так і обгортками над існуючими монолітами. Для інтеграції сервісів часто застосовується шина даних (Enterprise Service Bus). Архітектура SOA спрощує масштабування і модульність системи та стала основою для розвитку мікросервісів.

Мікросервісна архітектура

Мікросервісна архітектура описує систему як набір малих незалежних служб, кожна з яких реалізує окрему функцію та має власний життєвий цикл. Мікросервіси працюють у власних процесах і взаємодіють через легкі протоколи, такі як HTTP, WebSockets або AMQP. Кожен сервіс відповідає за конкретну частину бізнес-логіки й може бути розгорнутий окремо, що значно спрощує масштабування та розвиток складних систем. Такий підхід підвищує гнучкість, дозволяє використовувати різні технології в одному проєкті та спрощує підтримку великих розподілених додатків.

Хід Роботи

Діаграма класів

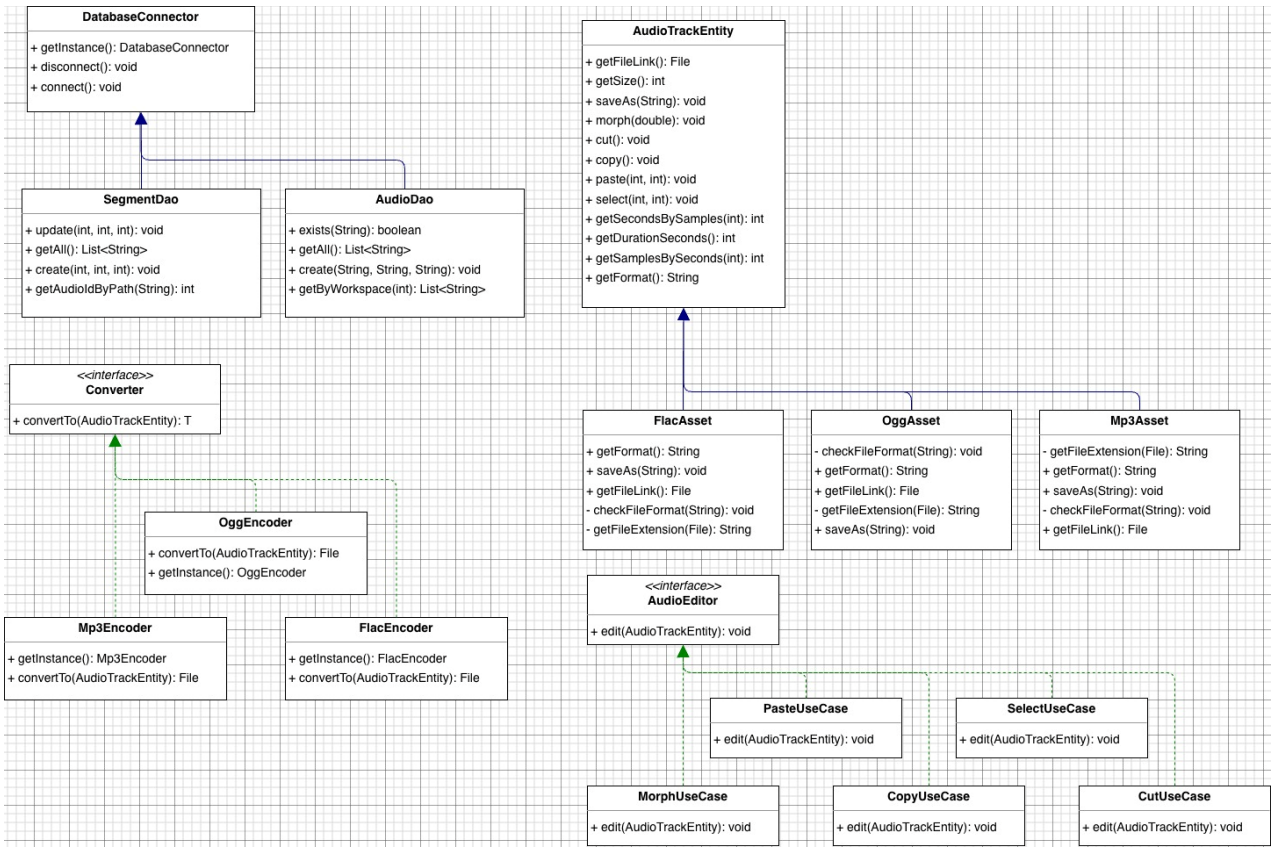

Рисунок 1 – Діаграма класів

На рисунку 1 накведено діаграму класів аудіоредактора, що відображає логічну структуру програмного забезпечення, основні класи, їхні атрибути, методи та взаємозв’язки між ними.

Центральним елементом системи є клас AudioTrackEntity, який описує сутність аудіотреку та містить методи для виконання основних операцій редагування cut(), copy(), paste(), morph(), а також методи для роботи з форматом файлу, отримання семплів і тривалості (getDurationSeconds(), getFormat(), getSampleBySeconds()).

Цей клас є базовим для роботи з різними типами звукових файлів і виступає ядром усієї системи.

Від AudioTrackEntity наслідуються класи Mp3Asset, OggAsset та FlacAsset, що реалізують підтримку різних аудіоформатів. Кожен із цих класів має власні методи для збереження (saveAs()), перевірки формату (checkFileFormat()), а також отримання розширення файлу (getFileExtension()), що дозволяє системі працювати з популярними форматами MP3, OGG та FLAC.

Окремий клас AudioEditor реалізує високорівневу логіку редагування файлів. Він містить метод edit(), який викликає відповідні операції залежно від сценарію редагування: копіювання, вставлення, вирізання або деформації звуку. Для реалізації цих сценаріїв використовуються окремі класи CutUseCase, CopyUseCase, SelectUseCase і MorphUseCase, кожен із яких виконує власну операцію редагування через метод edit(AudioTrackEntity).

Для кодування звукових доріжок у різні формати використовується абстрактний клас Converter<T>, який містить універсальний метод convertTo(AudioTrackEntity). Від нього наслідуються конкретні реалізації Mp3Encoder, OggEncoder та FlacEncoder, які забезпечують перетворення аудіо у відповідний формат.

Компонент DatabaseConnector відповідає за взаємодію з локальною базою даних SQLite. Він реалізує підключення, відключення та отримання екземпляра підключення (getInstance(), connect(), disconnect()). З ним працюють два DAO-класи AudioDao і SegmentDao, які реалізують доступ до даних аудіофайлів і сегментів.
AudioDao забезпечує методи create(), getAll(), exists() і getByWorkspace(), а SegmentDao: операції create(), update() та getAudioIdByPath(). Це дає змогу системі зберігати інформацію про проєкти, сегменти та аудіофайли у структурованому вигляді.

Діаграма класів демонструє, як система аудіоредактора розділена на логічні шари:

- рівень даних (DAO-класи, DatabaseConnector),
- рівень бізнес-логіки (AudioEditor, UseCase-класи),
- рівень моделей (AudioTrackEntity і похідні класи для форматів),
- рівень перетворень (Converter і Encoder-класи).

Фрагменти коду:

ClientRequest.java

```
package app.soundlab.client;

import java.util.Map;

public class ClientRequest {
    private String command;
    private Map<String, Object> parameters;

    public ClientRequest() {
    }

    public ClientRequest(String command, Map<String, Object> parameters) {
        this.command = command;
        this.parameters = parameters;
    }

    public String getCommand() {
        return command;
    }

    public void setCommand(String command) {
        this.command = command;
    }

    public Map<String, Object> getParameters() {
        return parameters;
```

```
    }

    public void setParameters(Map<String, Object> parameters) {
        this.parameters = parameters;
    }
}
```

TcpClient.java

```
package app.soundlab.client;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

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

public class TcpClient {
    private static final String DEFAULT_HOST = "localhost";
    private static final int DEFAULT_PORT = 8888;
    private static final Gson gson = new GsonBuilder().create();

    private final String host;
    private final int port;
    private Socket socket;
    private PrintWriter out;
    private BufferedReader in;

    public TcpClient() {
```

```
        this(DEFAULT_HOST, DEFAULT_PORT);
    }

    public TcpClient(String host, int port) {
        this.host = host;
        this.port = port;
    }

    public void connect() throws IOException {
        socket = new Socket(host, port);
        out = new PrintWriter(socket.getOutputStream(), true);
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }

    public ServerResponse sendRequest(String command, Map<String, Object>
parameters) throws IOException {
        if (socket == null || socket.isClosed()) {
            connect();
        }

        ClientRequest request = new ClientRequest(command, parameters);
        String requestJson = gson.toJson(request);

        out.println(requestJson.length());
        out.print(requestJson);
        out.print("\n");
        out.flush();

        String responseLengthStr = in.readLine();
```

```
        if (responseLengthStr == null) {
            throw new IOException("Server closed connection");
        }
        int responseLength = Integer.parseInt(responseLengthStr);

        char[] buffer = new char[responseLength];
        int totalRead = 0;
        while (totalRead < responseLength) {
            int read = in.read(buffer, totalRead, responseLength - totalRead);
            if (read == -1) {
                throw new IOException("Unexpected end of stream");
            }
            totalRead += read;
        }

        String responseJson = new String(buffer, 0, totalRead);
        return gson.fromJson(responseJson, ServerResponse.class);
    }

    public void disconnect() {
        try {
            if (in != null) in.close();
            if (out != null) out.close();
            if (socket != null && !socket.isClosed()) socket.close();
        } catch (IOException e) {
            System.err.println("Error closing connection: " + e.getMessage());
        }
    }
}
```

AudioEditorServer.java

```
    private ServerResponse handleLoadFile(String filePath) {
        try {
            if (filePath == null) {
                return new ServerResponse(false, "File path is required", null);
            }

            java.io.File file = new java.io.File(filePath);
            if (!file.exists()) {
                return new ServerResponse(false, "File does not exist: " + filePath,
null);
            }

            String fileName = file.getName().toLowerCase();
            if (!fileName.endsWith(".mp3") && !fileName.endsWith(".ogg") &&
!fileName.endsWith(".flac")) {
                return new ServerResponse(false, "Unsupported file format. Please
select MP3, OGG, or FLAC file.", null);
            }

            if (!audioDao.exists(filePath)) {
                String format = filePath.substring(filePath.lastIndexOf('.') + 1);
                String title = file.getName();
                audioDao.create(title, format, filePath);
            }

            selectedFilePath = filePath;
            currentTrack = resolveFor(file);

            Map<String, Object> data = new HashMap<>();
```

```
            data.put("fileName", file.getName());
            data.put("duration", currentTrack.getDurationSeconds());

            return new ServerResponse(true, "File loaded successfully", data);
        } catch (Exception e) {
            return new ServerResponse(false, "Failed to load file: " + e.getMessage(),
null);
        }
    }
    private ServerResponse handleConvert(String format) {
        try {
            if (currentTrack == null) {
                return new ServerResponse(false, "Please load a file first.", null);
            }

            java.io.File convertedFile;
            switch (format) {
                case "mp3" -> convertedFile =
Mp3Encoder.get().encode(currentTrack);
                case "ogg" -> convertedFile = OggEncoder.get().encode(currentTrack);
                case "flac" -> convertedFile = FlacEncoder.get().encode(currentTrack);
                default -> throw new IllegalArgumentException("Unsupported format:
" + format);
            }

            Map<String, Object> data = new HashMap<>();
            data.put("convertedFilePath", convertedFile.getAbsolutePath());
            data.put("fileName", convertedFile.getName());
```

```
            return new ServerResponse(true, "File converted to " +
format.toUpperCase() + " successfully!", data);
        } catch (Exception e) {
            return new ServerResponse(false, "Error during conversion: " +
e.getMessage(), null);
        }
    }

    private ServerResponse handleCut(int start, int end) {
        try {
            if (currentTrack == null) {
                return new ServerResponse(false, "Please load a file first.", null);
            }

            int trackDuration = currentTrack.getDurationSeconds();
            if (!isSegmentValid(start, end, trackDuration)) {
                return new ServerResponse(false, "Invalid segment bounds. Ensure 0
<= Start < End <= " + trackDuration, null);
            }

            int startSamples = currentTrack.getSamplesBySeconds(start);
            int endSamples = currentTrack.getSamplesBySeconds(end);
            currentTrack.select(startSamples, endSamples);
            currentTrack.cut();

            return new ServerResponse(true, "Segment cut successfully.", null);
        } catch (Exception e) {
            return new ServerResponse(false, "Error cutting segment: " +
e.getMessage(), null);
        }
```

```
    }

    private ServerResponse handleCopy(int start, int end) {
        try {
            if (currentTrack == null) {
                return new ServerResponse(false, "Please load a file first.", null);
            }

            int trackDuration = currentTrack.getDurationSeconds();
            if (!isSegmentValid(start, end, trackDuration)) {
                return new ServerResponse(false, "Invalid segment bounds. Ensure 0
<= Start < End <= " + trackDuration + ".", null);
            }

            int startSamples = currentTrack.getSamplesBySeconds(start);
            int endSamples = currentTrack.getSamplesBySeconds(end);
            currentTrack.select(startSamples, endSamples);
            currentTrack.copy();

            return new ServerResponse(true, "Segment copied successfully.", null);
        } catch (Exception e) {
            return new ServerResponse(false, "Error copying segment: " +
e.getMessage(), null);
        }
    }

    private ServerResponse handlePaste(int position) {
        try {
            if (currentTrack == null) {
```

```
                return new ServerResponse(false, "Please load a file first.", null);
            }

            int samplesPerSecond = currentTrack.getSamplesBySeconds(1);
            currentTrack.paste(position, samplesPerSecond);

            return new ServerResponse(true, "Segment pasted successfully.", null);
        } catch (Exception e) {
            return new ServerResponse(false, "Error pasting segment: " +
e.getMessage(), null);
        }
    }

    private ServerResponse handleMorph(int start, int end, double factor) {
        try {
            if (currentTrack == null) {
                return new ServerResponse(false, "Please load a file first.", null);
            }
            int startSamples = currentTrack.getSamplesBySeconds(start);
            int endSamples = currentTrack.getSamplesBySeconds(end);
            currentTrack.select(startSamples, endSamples);
            currentTrack.morph(factor);

            return new ServerResponse(true, "Segment deformed successfully.", null);
        } catch (Exception e) {
            return new ServerResponse(false, "Error deforming segment: " +
e.getMessage(), null);
        }
    }
```

Вихідний код:
https://github.com/mandarinchik21/AudioEditor/tree/main/lab9

Висновок: У ході виконання роботи було розглянуто різні архітектурні підходи до побудови розподілених систем, зокрема клієнт-серверну, однорангову, сервіс-орієнтовану та мікросервісну архітектури. Кожен із цих підходів має власні переваги, принципи та сценарії застосування, що дозволяє обирати оптимальну модель залежно від вимог системи. Порівняння показало, що сучасні програмні рішення рухаються у напрямку децентралізації та підвищення гнучкості, а мікросервіси та SOA забезпечують найкращу масштабованість і незалежність компонентів. Отримані теоретичні знання формують розуміння того, як будувати ефективні, стійкі та керовані системи, а також як правильно розподіляти логіку між клієнтом, сервером та сервісами.

Контрольні запитання

1. Що таке клієнт-серверна архітектура?
Це модель, у якій є клієнти, що надсилають запити, і сервери, що їх обробляють. Клієнт відповідає за інтерфейс та взаємодію з користувачем, сервер за зберігання та обробку даних.

2. Розкажіть про сервіс-орієнтовану архітектуру.
Сервіс-орієнтована архітектура (SOA) це підхід, де система складається з незалежних сервісів з чіткими інтерфейсами, які взаємодіють між собою через стандартизовані протоколи. Кожен сервіс виконує окрему бізнес-функцію.

3. Якими принципами керується SOA?
• слабка зв’язаність між сервісами
• стандартизовані інтерфейси
• незалежне оновлення та розгортання сервісів
• повторне використання компонентів
• взаємодія через повідомлення, а не через спільні бази даних

4. Як між собою взаємодіють сервіси в SOA?
Сервіси взаємодіють через обмін повідомленнями (HTTP, SOAP, REST). Вони не звертаються до внутрішніх структур один одного, а працюють тільки через контракт- публічний інтерфейс.

5. Як розробники взнають про існуючі сервіси і як робити до них запити?
Сервіси реєструються у сервісному реєстрі (каталозі). Розробники знаходять

потрібний сервіс у цьому каталозі та використовують його опис (WSDL, OpenAPI/Swagger) для виконання запитів.

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

Переваги:

• централізоване зберігання даних

• просте масштабування сервера

• легке оновлення, достатньо оновити сервер

• безпечніше, ніж P2P

Недоліки:

• залежність від сервера (точка відмови)

• сервер може бути перевантажений

• потрібні потужні серверні ресурси

7. У чому полягають переваги та недоліки однорангової моделі (P2P)?

Переваги:

• децентралізація

• висока стійкість до відмов

• ресурсами ділиться вся мережа

Недоліки:

• складніше забезпечити безпеку

• важче синхронізувати дані

• пошук ресурсів може бути повільним

8. Що таке мікро-сервісна архітектура?

Це підхід, де система складається з дрібних незалежних сервісів, кожен з яких

виконує одну конкретну бізнес-функцію. Кожен мікросервіс розгортається окремо та взаємодіє з іншими через легкі протоколи.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?
• HTTP / HTTPS
• REST
• WebSockets
• AMQP (RabbitMQ)
• gRPC
• Kafka (як стрімінг-протокол)

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли між веб-контролерами та DAO додається шар сервісів?
Ні. Це звичайний шар бізнес-логіки всередині моноліту. SOA передбачає розподілені автономні сервіси, які можна розгортати окремо. В монолітній архітектурі всі ці "сервіси" лише частини одного додатку, а не незалежні компоненти.