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 №3

із дисципліни «*Технології розроблення програмного забезпечення*»

Основи проектування розгортання.

Аудіоредактор

Виконала
студентка групи ІА–34
Кузьменко В.С.

Перевірив
викладач
Мягкий М.Ю.

Київ-2025

ЗМІСТ

Тема: Основи проектування розгортання.

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторніроботі.

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроєктувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.

• Розробити діаграму компонентів для проєктованої системи.

• Розробити діаграму розгортання для проєктованої системи.

• Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.

• На основі спроєктованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми. В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та відображенням на UI).

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму розгортання з описом, діаграму компонентів системи з описом, діаграми послідовностей, а також вихідний код системи, який було додано в цій лабораторній роботі.

Тема роботи:

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

Діаграма розгортання (Deployment Diagram)

Діаграми розгортання показують фізичне розташування елементів системи, демонструючи, на якому обладнанні працює програмне забезпечення. Головні елементи діаграми — вузли, які пов’язані між собою інформаційними шляхами. Вузли бувають двох типів: пристрої (фізичне обладнання, як-от комп’ютери) і середовище виконання (програмне забезпечення, яке може включати інше). В узах можна деталізувати артефакти, наприклад, компоненти чи класи. Проте така деталізація рідко потрібна, оскільки вона може змістити фокус із розгортання системи на її структуру.

Діаграма компонентів (Component Diagram)

Ця діаграма показує систему, розбиту на модулі. Є три типи діаграм компонентів: логічні, фізичні й виконавчі. Найчастіше використовують логічне розбиття, коли система уявляється як набір автономних модулів, які можуть взаємодіяти між собою. Наприклад, система продажів може складатися з каси, черги повідомлень, сервера продажів і системи обліку. Компоненти можуть належати до різних фізичних вузлів, проте взаємозамінність робить їх гнучкими для клієнтів. У фізичному поділі кожен компонент може бути розташований на окремому сервері чи комп’ютері, але такий підхід застарів і зазвичай замінюється діаграмами розгортання. Виконавчі діаграми описують компоненти як файли, наприклад, .exe чи бази даних.

Діаграма діяльності (Activity Diagram)

Цей тип діаграм моделює виконання операцій, їх логіку і порядок переходів між діями. Діяльність — це набір обчислень, які приводять до певного результату. Графічно вона схожа на діаграми станів, але кожне її стан — це виконання конкретної операції. Дія, завершуючись, передає керування наступному стану. Стан 4 дії зображується прямокутником із закругленими кутами, всередині якого вказується унікальне ім’я дії. Якщо дія складна, вона

може бути представлена як під діяльність, яка позначається спеціальною піктограмою.

Діаграма послідовності (Sequence Diagram)

Ця діаграма показує взаємодію об’єктів у часі. На вертикальній осі розташовані лінії життя об’єктів, які позначають час їх існування в системі. Ініціатор взаємодії знаходиться зліва. Об’єкти можуть бути створені або знищені в будь-який момент. Знищення позначається символом "X". Лінії взаємодії між об’єктами відображають обмін повідомленнями, їх порядок і час виконання.

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Спроєктувати діаграми для системи: розгортання, компонентів та дві послідовності.
3. Доопрацювати програмну частину системи, додавши мінімум дві форми та забезпечивши повний цикл роботи з даними.
4. Скласти звіт про виконану роботу.

Діаграма розгортання

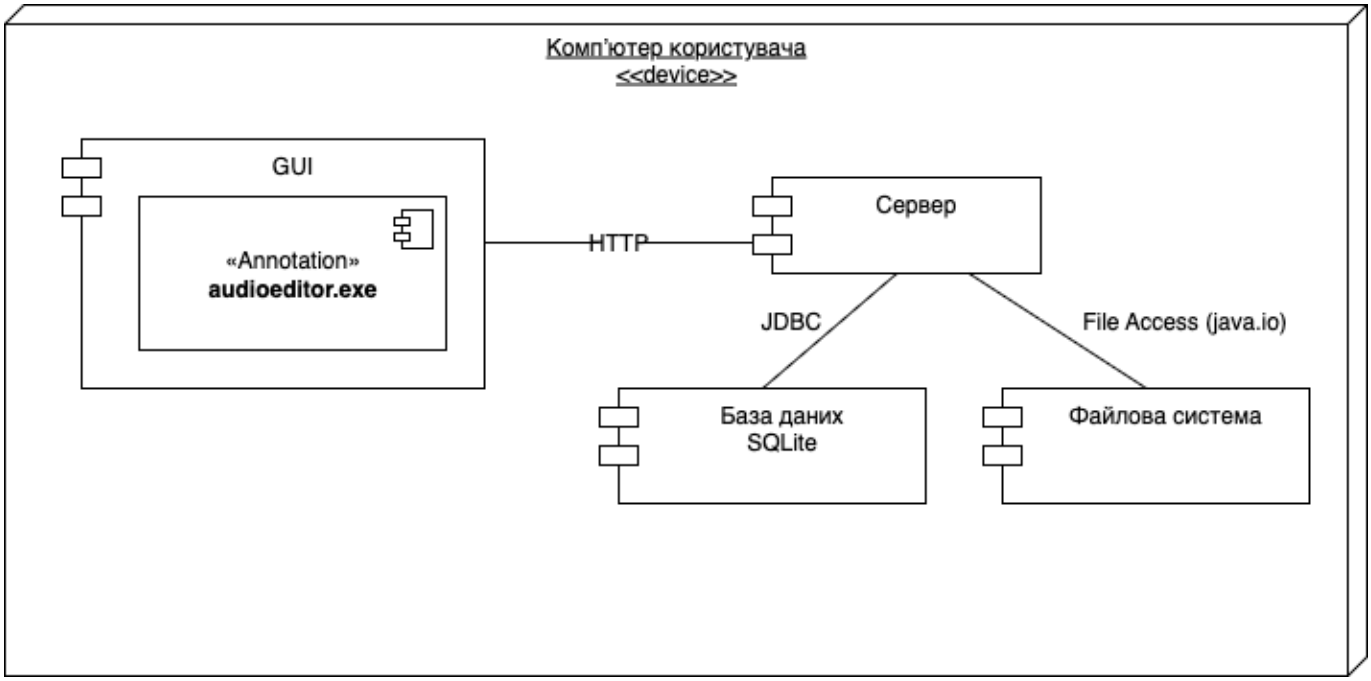

Рис. 1 Діаграма розгортання

Діаграма розгортання відображає структуру архітектури аудіоредактора, який функціонує на локальному пристрої користувача. У центрі архітектури знаходиться основний пристрій - «Комп'ютер користувача» (позначений як вузол <>), на якому безпосередньо розгорнуто програмні компоненти редактора.

1. Графічний інтерфейс користувача (GUI)

На комп’ютері користувача знаходиться графічний інтерфейс (GUI), що реалізований у вигляді виконуваного файлу аудіоредактора. Саме цей компонент забезпечує користувачеві доступ до функціоналу програми:

- надає інтерфейс для взаємодії (вікна, меню, кнопки, інструменти редагування аудіо);
- приймає дії користувача (завантаження аудіофайлу, застосування ефектів, збереження проекту тощо);
- надсилає запити до серверної частини для виконання більш складних операцій.

2. Серверна частина

Усередині того ж пристрою працює компонент «Сервер», який виконує роль ядра системи. Його завдання:

- обробка запитів від GUI;
- реалізація бізнес-логіки (наприклад, застосування фільтрів, мікшування треків, експорт у різні формати);
- управління даними користувача.

3. Ресурси серверної частини

Сервер взаємодіє з двома основними типами ресурсів:

1. База даних SQLite
    - використовується для зберігання структурованих даних (наприклад, налаштувань користувача, історії проектів, метаданих аудіо);
    - доступ до БД здійснюється через JDBC-драйвер, який забезпечує стандартний інтерфейс взаємодії між сервером і системою керування базою даних.
2. Файлова система комп’ютера

- забезпечує фізичне зберігання та доступ до аудіофайлів (наприклад, імпортованих записів або готових проектів);
- сервер працює з файлами через стандартні механізми файлового вводу/виводу (читання, запис, редагування).

Таким чином, діаграма демонструє взаємозв’язки між усіма ключовими компонентами:

- користувач працює з GUI;
- GUI взаємодіє із Сервером;
- Сервер у свою чергу має доступ як до бази даних, так і до файлової системи, забезпечуючи повний цикл роботи аудіоредактора.

Діаграма компонентів

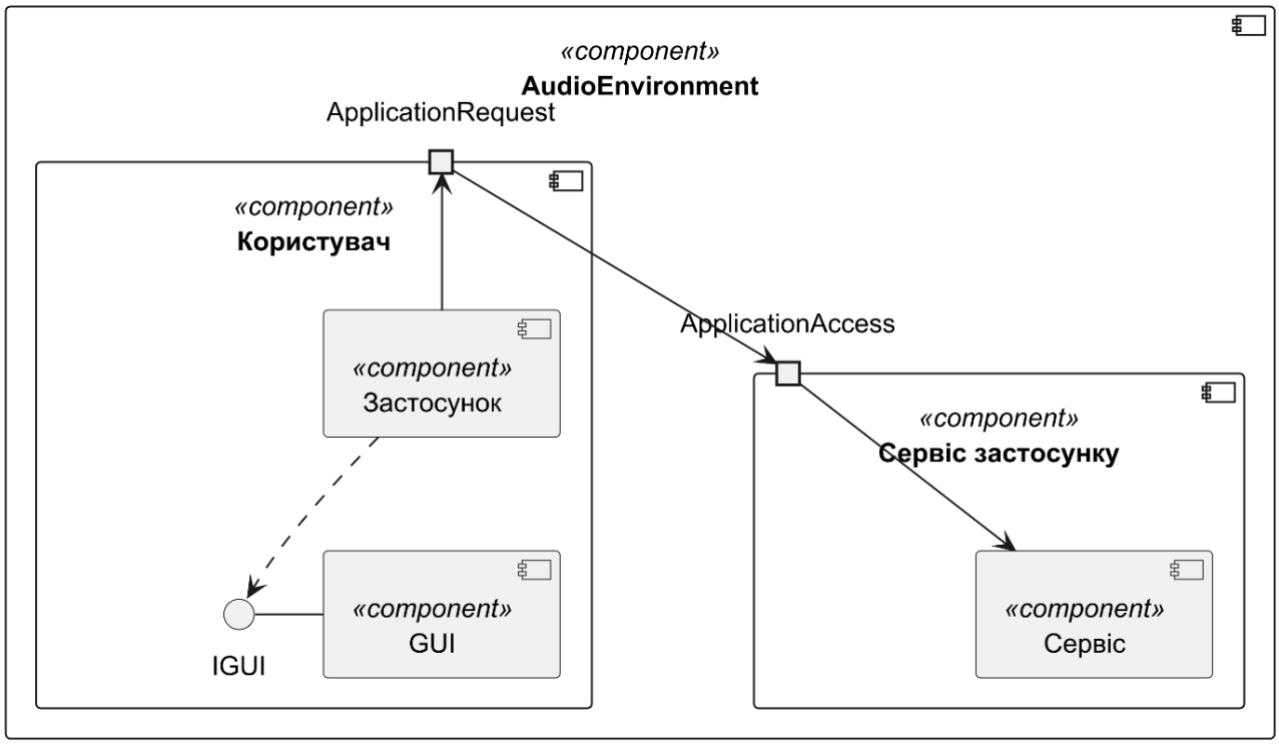

Рис. 2 Діаграма компонентів

Діаграма компонентів демонструє архітектурну структуру аудіоредактора з точки зору організації програмних компонентів та взаємодії між ними. Вона відображає, які компоненти входять до складу системи, які інтерфейси вони реалізують та яким чином взаємодіють один з одним.

У межах загального компонента AudioEnvironment можна виділити три ключові складові:

1. Користувач (User)
2. Застосунок (Application)
3. Сервіс застосунку (Service)

1. Користувач

Кінцевий користувач є зовнішнім учасником системи. Його основна роль полягає у взаємодії з аудіоредактором через графічний інтерфейс користувача (GUI).

- Користувач подає вхідні дані (наприклад, відкриття аудіофайлу, застосування ефектів, збереження проєкту).
- Отримує результати у вигляді візуального представлення та обробленого аудіо.

2. Графічний інтерфейс користувача (GUI)

Компонент GUI інкапсулює весь графічний інтерфейс програми:

- забезпечує відображення елементів управління (вікна, меню, панелі інструментів, таймлайн для редагування аудіо);
- надає користувачеві можливість взаємодії з системою через стандартні елементи управління;
- реалізує інтерфейс IGUI, що описує доступний набір функцій для взаємодії із застосунком.

Таким чином, GUI виступає проміжною ланкою між користувачем і внутрішньою логікою програми.

3. Застосунок (Application)

Компонент Застосунок реалізує основний функціонал аудіоредактора та взаємодіє з іншими частинами системи:

- Через інтерфейс IGUI застосунок отримує дані від GUI (команди користувача) і може ініціювати оновлення інтерфейсу.
- Для доступу до ресурсів і логіки системи застосунок використовує механізм ApplicationRequest, який слугує каналом для передачі запитів до сервісної частини.
- У застосунку зосереджена логіка керування процесом обробки аудіо (наприклад, виклик методів сервісу для обробки сигналу, генерації ефектів, мікшування треків).

4. Сервісна частина (Service)

У складі системи виділено компонент Сервіс, який є внутрішнім модулем, відповідальним за виконання операцій, що надходять від застосунку.

- Сервіс реалізує бізнес-логіку: обробку аудіосигналів, виконання математичних операцій, керування ресурсами.
- Доступ до цього компонента здійснюється через інтерфейс ApplicationAccess, що дозволяє застосунку викликати відповідні методи й отримувати результати.
- Завдяки цьому забезпечується чітке розділення відповідальності: застосунок відповідає за координацію процесів, а сервіс — за виконання конкретних обчислювальних завдань.

Підсумовуючи вищезазначене:

- Користувач взаємодіє з GUI
- GUI <-> Застосунок (через інтерфейс IGUI)
- Застосунок -> надсилає запити до Сервісу (через ApplicationRequest / ApplicationAccess)
- Сервіс -> виконує необхідні операції й повертає результат у Застосунок, який оновлює GUI.

Таким чином, забезпечується повний цикл взаємодії - від дій користувача до виконання внутрішньої логіки системи та відображення результатів.

Діаграма послідовностей

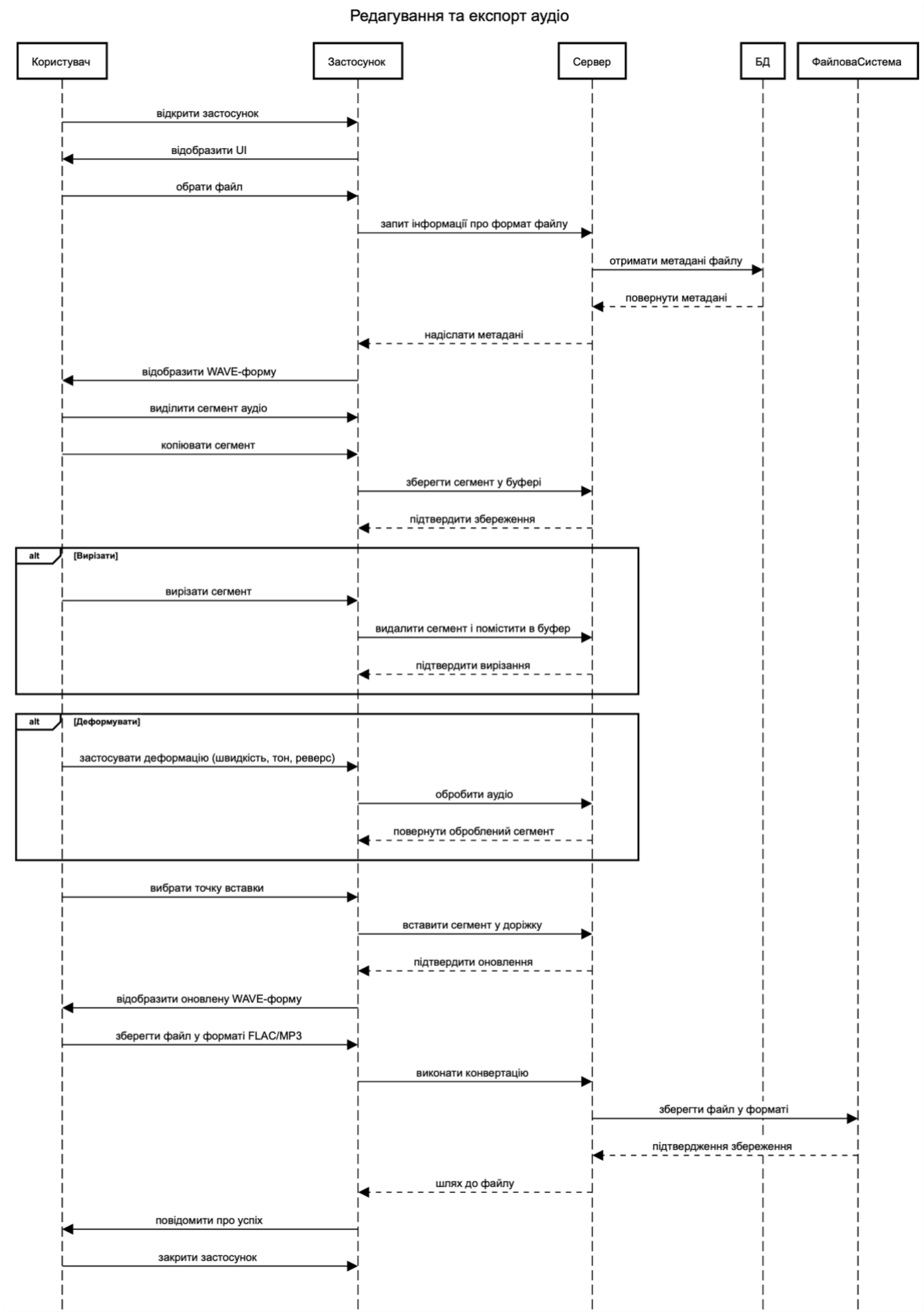

Рис. 3.1 Діаграма послідовностей. Редагування та експорт аудіо

Діаграма послідовностей відображає покроковий процес взаємодії користувача з аудіоредактором під час редагування та експорту файлу. Вона демонструє динамічну взаємодію між основними учасниками: Користувачем, Застосунком (Application), Сервером (Server), Базою даних (Database) та Файловою системою.

1. **Запуск застосунку**

- Користувач відкриває аудіоредактор.
- Застосунок ініціалізується і відображає графічний інтерфейс (GUI).

2. **Завантаження аудіофайлу**

- Користувач обирає аудіофайл для редагування.
- Застосунок надсилає запит на сервер для отримання інформації про файл.
- Сервер звертається до бази даних, щоб перевірити наявність метаданих.
- База даних повертає дані на сервер, після чого сервер передає їх у застосунок.
- Застосунок будує та відображає WAVE-форму аудіофайлу.

3. **Редагування аудіо**

- Користувач може виділяти сегменти та виконувати дії: копіювання, вставка, вирізання.
- Вибрані зміни обробляються сервером та відображаються у GUI у вигляді оновленої WAVE-форми.

4. **Збереження у новому форматі**

- Користувач ініціює збереження результату (наприклад, у форматі FLAC чи MP3).
- Застосунок надсилає файл на сервер.

- Сервер виконує конвертацію та зберігає результат у файловій системі.
- Сервер передає шлях до збереженого файлу назад у застосунок.
- Користувач отримує повідомлення про успішне збереження.

5. Завершення роботи

- Користувач закриває програму.
- Система завершує роботу.

Таким чином, діаграма демонструє повний цикл: від запуску програми та завантаження аудіо до його редагування і експорту у новий формат.

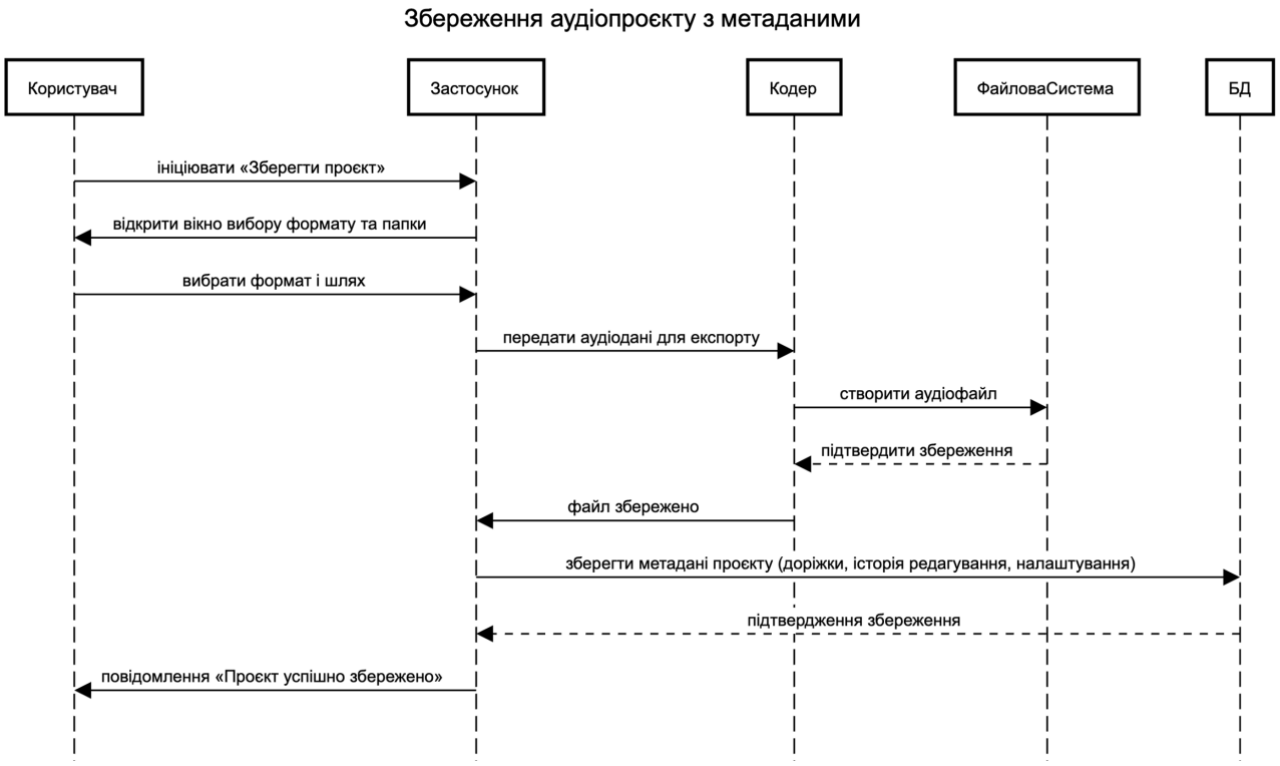

Рис. 3.2 Діаграма послідовностей. Збереження з метаданими

Дана діаграма деталізує процес збереження результатів роботи користувача у вигляді готового аудіофайлу та проєкту з метаданими. Вона описує взаємодію між Користувачем, Застосунком (Application), Модулем кодування (Encoder), Файловою системою та Базою даних (Database).

1. **Ініціація збереження**

- Користувач запускає команду «Зберегти проєкт».
- Застосунок відкриває вікно вибору формату та місця збереження.
- Користувач обирає цільовий формат (наприклад, .ogg, .flac, .mp3) та папку.

2. **Експорт аудіофайлу**

- Застосунок передає аудіодані модулю кодування.
- Модуль кодування створює файл у вибраному форматі у файловій системі.
- Файлова система підтверджує успішне збереження.
- Модуль кодування повідомляє застосунок про завершення експорту.

3. **Збереження проєкту з метаданими**

- Застосунок передає у базу даних метадані проєкту: структуру доріжок, параметри ефектів, історію редагувань.
- База даних підтверджує збереження.

4. **Завершення операції**

- Застосунок повідомляє користувача: «Проєкт успішно збережено».

Таким чином, діаграма демонструє не лише створення готового аудіофайлу, але й паралельне збереження проєкту, що дозволяє у майбутньому повторно відкривати його для редагування.

Код системи

Лістинг 1- AudioProcessor

```
package org.audioeditor.audio;

import javax.sound.sampled.*;
import java.io.File;
import java.io.IOException;
```

```
public class AudioProcessor {
    private File audioFile;

    public AudioProcessor(File audioFile) {
        this.audioFile = audioFile;
    }

    public int[] getWaveform() throws IOException, UnsupportedAudioFileException
{
        AudioInputStream audioInputStream =
AudioSystem.getAudioInputStream(audioFile);
        AudioFormat format = audioInputStream.getFormat();
        byte[] audioBytes = audioInputStream.readAllBytes();

        // Підтримка лише 16-бітного PCM
        if (format.getSampleSizeInBits() != 16 || format.isBigEndian()) {
            throw new UnsupportedAudioFileException("Підтримуються лише 16-бітні
PCM файли!");
        }

        int[] samples = new int[audioBytes.length / 2];
        for (int i = 0; i < samples.length; i++) {
            int low = audioBytes[2 * i] & 0xFF;
            int high = audioBytes[2 * i + 1];
            samples[i] = (high << 8) | low;
        }

        return samples;
    }
}
```

Лістинг 2- Audio Repository

```
package org.audioeditor.repository;

import org.audioeditor.database.DatabaseConnection;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class AudioRepository {
    public void addAudio(String name, String format, String path) {
        String sql = "INSERT INTO Audio (name, format, path) VALUES (?, ?, ?)";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setString(1, name);
            statement.setString(2, format);
            statement.setString(3, path);
            statement.executeUpdate();
        } catch (SQLException e) {
            throw new RuntimeException("Failed to insert audio", e);
        }
    }

    public List<String> getAllAudio() {
        String sql = "SELECT * FROM Audio";
        List<String> audioList = new ArrayList<>();
```

```
        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql);
             ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                audioList.add(resultSet.getString("name"));
            }
        } catch (SQLException e) {
            throw new RuntimeException("Failed to fetch audio", e);
        }

        return audioList;
    }

    public List<String> getAudioByProject(int projectId) {
        String sql = "SELECT Audio.name FROM Audio " +
                "JOIN Project_Audio ON Audio.id = Project_Audio.audio_id " +
                "WHERE Project_Audio.project_id = ?";
        List<String> audioList = new ArrayList<>();

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, projectId);
            try (ResultSet resultSet = statement.executeQuery()) {
                while (resultSet.next()) {
                    audioList.add(resultSet.getString("name"));
                }
            }
        } catch (SQLException e) {
            throw new RuntimeException("Failed to fetch audio for project", e);
        }

        return audioList;
    }
}
```

Лістинг 3- ProjectRepository

```
package org.audioeditor.repository;

import org.audioeditor.database.DatabaseConnection;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class ProjectRepository {
    public void addProject(String name) {
        String sql = "INSERT INTO Project (name) VALUES (?)";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setString(1, name);
            statement.executeUpdate();
        } catch (SQLException e) {
            throw new RuntimeException("Failed to insert project", e);
        }
    }
```

```
    public List<String> getAllProjects() {
        String sql = "SELECT * FROM Project";
        List<String> projectList = new ArrayList<>();

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql);
             ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                projectList.add("Project ID: " + resultSet.getInt("id") +
                        ", Name: " + resultSet.getString("name") +
                        ", Created At: " + resultSet.getString("created_at"));
            }
        } catch (SQLException e) {
            throw new RuntimeException("Failed to fetch projects", e);
        }

        return projectList;
    }

    public void addAudioToProject(int projectId, int audioId) {
        String sql = "INSERT INTO Project_Audio (project_id, audio_id) VALUES
(?, ?)";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, projectId);
            statement.setInt(2, audioId);
            statement.executeUpdate();
        } catch (SQLException e) {
            throw new RuntimeException("Failed to add audio to project", e);
        }
    }
}
```

Лістинг 4- TrackRepository

```
package org.audioeditor.repository;

import org.audioeditor.database.DatabaseConnection;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class TrackRepository {
    public void addTrack(int audioId, int startTime, int endTime) {
        String sql = "INSERT INTO Track (audio_id, start_time, end_time) VALUES
(?, ?, ?)";

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, audioId);
            statement.setInt(2, startTime);
            statement.setInt(3, endTime);
            statement.executeUpdate();
        } catch (SQLException e) {
            throw new RuntimeException("Failed to insert track", e);
        }
    }
```

```
    public List<String> getAllTracks() {
        String sql = "SELECT * FROM Track";
        List<String> trackList = new ArrayList<>();

        try (Connection connection = DatabaseConnection.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql);
             ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                trackList.add("Track ID: " + resultSet.getInt("id") +
                        ", Audio ID: " + resultSet.getInt("audio_id") +
                        ", Start Time: " + resultSet.getInt("start_time") +
                        ", End Time: " + resultSet.getInt("end_time"));
            }
        } catch (SQLException e) {
            throw new RuntimeException("Failed to fetch tracks", e);
        }

        return trackList;
    }
}
```

Лістинг 5- AudioConverter

```
package org.audioeditor.service;

import org.audioeditor.audiotrack.Audiotrack;
import org.audioeditor.converter.Converter;

public class AudioConverter {
    Audiotrack audiotrack;
    Converter converter;
}
```

Лістинг 6- AudioEditor

```
package org.audioeditor.service;

import org.audioeditor.audiotrack.Audiotrack;
import org.audioeditor.editor.Editor;

public class AudioEditor {
    Audiotrack audiotrack;
    Editor selector;
    Editor copier;
    Editor paste;
    Editor cut;

    Audiotrack select(Audiotrack audiotrack, int length) {
        return null;
    }

    Audiotrack copy(Audiotrack audiotrack) {
        return null;
    }

    Audiotrack paste(Audiotrack audiotrack) {
        return null;
    }
```

```
    void cut(Audiotrack audiotrack) {

    }
}
```

Лістинг 7- AudioOutput

```
package org.audioeditor.service;

import org.audioeditor.audiotrack.Audiotrack;

public class AudioOutput {
    Audiotrack audiotrack;

    void outputWaveForm(Audiotrack audiotrack) {

    }
}
```

Лістинг 8- Main

```
package org.audioeditor;

import org.audioeditor.audiotrack.Flac;
import org.audioeditor.database.DatabaseInitializer;

public class Main {
    public static void main(String[] args) {
        DatabaseInitializer.initializeDatabase();
        new AudioEditorUI();
    }
}
```

Лістинг 9- AudioEditorUI

```
package org.audioeditor;

import org.audioeditor.repository.AudioRepository;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

public class AudioEditorUI {
    private JFrame frame;
    private AudioRepository repository;

    public AudioEditorUI() {
        repository = new AudioRepository();
        frame = new JFrame("Audio Editor");
        frame.setSize(500, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(null);

        // Кнопка для вибору файлу
        JButton addButton = new JButton("Add Audio");
        addButton.setBounds(50, 50, 150, 30);
```

```
        frame.add(addButton);

        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JFileChooser fileChooser = new JFileChooser();
                fileChooser.setDialogTitle("Select Audio File");
                fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
                int result = fileChooser.showOpenDialog(frame);
                if (result == JFileChooser.APPROVE_OPTION) {
                    File selectedFile = fileChooser.getSelectedFile();
                    // Додаємо файл у репозиторій
                    repository.addAudio(selectedFile.getName(),
                            getFileExtension(selectedFile),
                            selectedFile.getAbsolutePath());
                    JOptionPane.showMessageDialog(frame, "Audio added: " +
selectedFile.getName());
                } else {
                    JOptionPane.showMessageDialog(frame, "No file selected");
                }
            }
        });

        // Кнопка для перегляду аудіо
        JButton viewButton = new JButton("View Audios");
        viewButton.setBounds(50, 100, 150, 30);
        frame.add(viewButton);

        viewButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String audios = String.join(", ", repository.getAllAudio());
                if (audios.isEmpty()) {
                    audios = "No audio files added yet.";
                }
                JOptionPane.showMessageDialog(frame, "Audios: " + audios);
            }
        });

        frame.setVisible(true);
    }

    // Допоміжний метод для отримання розширення файлу
    private String getFileExtension(File file) {
        String name = file.getName();
        int lastIndex = name.lastIndexOf('.');
        if (lastIndex > 0 && lastIndex < name.length() - 1) {
            return name.substring(lastIndex + 1);
        }
        return "";
    }

    public static void main(String[] args) {
        new AudioEditorUI();
    }
}
```

Лістинг 10- MainApp

```
package org.audioeditor.app;

import org.audioeditor.audio.AudioProcessor;
```

```
import org.audioeditor.ui.WaveformCanvas;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import javax.sound.sampled.UnsupportedAudioFileException;
import java.io.File;
import java.io.IOException;

import javafx.scene.control.ToolBar;
import javafx.scene.control.Button;

public class MainApp extends Application {
    @Override
    public void start(Stage primaryStage) {
        VBox root = new VBox(10);

        // Кнопка для вибору файлу
        Button selectFileButton = new Button("Обрати аудіофайл");
        selectFileButton.setOnAction(e -> {
            FileChooser fileChooser = new FileChooser();
            fileChooser.getExtensionFilters().addAll(
                    new FileChooser.ExtensionFilter("WAV Files", "*.wav"),
                    new FileChooser.ExtensionFilter("MP3 Files", "*.mp3"),
                    new FileChooser.ExtensionFilter("FLAC Files", "*.flac"),
                    new FileChooser.ExtensionFilter("Усі підтримувані", "*.wav",
"*.mp3", "*.flac"));
            File selectedFile = fileChooser.showOpenDialog(primaryStage);

            if (selectedFile != null) {
                try {
                    AudioProcessor processor = new AudioProcessor(selectedFile);
                    int[] waveform = processor.getWaveform();

                    // Створюємо Canvas для відображення WAVE-форми
                    WaveformCanvas waveformCanvas = new WaveformCanvas(waveform,
800, 400);

                    // Додаємо Canvas у кореневий контейнер
                    root.getChildren().add(waveformCanvas);
                } catch (IOException | UnsupportedAudioFileException ex) {
                    System.err.println("Помилка обробки аудіофайлу: " +
ex.getMessage());
                }
            }
        });

        root.getChildren().add(selectFileButton);

        Scene scene = new Scene(root, 900, 600);
        primaryStage.setTitle("Аудіоредактор");
        primaryStage.setScene(scene);
        primaryStage.show();

        // Створюємо панель інструментів
        ToolBar toolBar = new ToolBar();

        Button copyButton = new Button("Копіювати");
        Button pasteButton = new Button("Вставити");
        Button cutButton = new Button("Вирізати");

        // Додаємо обробку подій для кнопок
```

```
        copyButton.setOnAction(e -> {
            // TODO: Реалізувати копіювання обраного сегменту
            System.out.println("Копіювати сегмент");
        });

        pasteButton.setOnAction(e -> {
            // TODO: Реалізувати вставку сегменту
            System.out.println("Вставити сегмент");
        });

        cutButton.setOnAction(e -> {
            // TODO: Реалізувати вирізання сегменту
            System.out.println("Вирізати сегмент");
        });

        // Додаємо кнопки в панель інструментів
        toolBar.getItems().addAll(copyButton, pasteButton, cutButton);
        root.getChildren().add(0, toolBar);
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Висновок

У процесі виконання роботи було проведено аналіз основних теоретичних засад, що стосуються створення UML-діаграм, а також здійснено їх практичну розробку для моделювання архітектури аудіоредактора.

У ході роботи виконано такі завдання:

1. Створено діаграму розгортання, яка відобразила фізичну архітектуру системи, показавши, як програмні компоненти розподілені між апаратними вузлами та яким чином відбувається взаємодія між ними.
2. Розроблено діаграму компонентів, що описує логічну структуру системи, її модулі та взаємозв’язки між ними.
3. Побудовано діаграму послідовностей, яка деталізує порядок обміну повідомленнями та взаємодію між користувачем, застосунком і сервісними складовими під час виконання ключових сценаріїв.
4. Сформовано звіт, у якому узагальнено результати роботи, представлено створені UML-моделі та пояснено їх практичне призначення.

Отримані результати мають низку важливих переваг:

- наочно відображено ключові аспекти архітектури системи, що дозволяє чіткіше розуміти її структуру та функціонування;
- побудовані діаграми можуть використовуватися як для документації проекту, так і для комунікації між розробниками, тестувальниками та іншими учасниками команди;
- створені UML-моделі можуть стати основою для подальшого вдосконалення та впровадження системи, забезпечуючи узгодженість між етапами розробки.

Таким чином, виконана робота сприяла не лише поглибленню теоретичних знань у галузі моделювання інформаційних систем, але й дала практичні навички застосування UML для проектування архітектури. Отримані результати можуть

бути корисними на етапах розробки, тестування та впровадження програмного забезпечення, а також забезпечують фундамент для подальшої еволюції системи.

Відповіді на теоретичні питання

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) - це структурна UML-діаграма, яка відображає фізичну архітектуру системи. Вона показує, на яких апаратних вузлах (комп'ютерах, серверах, пристроях) розгорнуті програмні компоненти, а також яким чином ці вузли взаємодіють між собою через мережеві або інші з'єднання.

2. Які бувають види вузлів на діаграмі розгортання?

- Апаратні вузли (наприклад, сервер, комп'ютер, мобільний пристрій, маршрутизатор);
- Програмні вузли (віртуальні машини, середовища виконання, контейнерні сервіси);
- Вкладені вузли (наприклад, сервер додатків усередині фізичного сервера).

3. Які бувають зв'язки на діаграмі розгортання?

- Асоціації/комунікаційні зв'язки (показують канали обміну повідомленнями між вузлами);
- Залежності (показують, що один елемент залежить від іншого);
- Розміщення (Deployment) - відображають, який компонент або артефакт розгорнуто на конкретному вузлі.

4. Які елементи присутні на діаграмі компонентів?

- Компоненти (модулі або частини програмного забезпечення);
- Інтерфейси (надавані й потрібні, які описують способи взаємодії);
- Залежності між компонентами;

- Артефакти (файли, бібліотеки, виконувані модулі, які реалізують компоненти).

5. Що становлять собою зв’язки на діаграмі компонентів?

Зв’язки на діаграмі компонентів описують залежності та взаємодію між компонентами. Вони показують, які інтерфейси один компонент реалізує, а інший використовує, а також як модулі співпрацюють для забезпечення роботи системи.

6. Які бувають види діаграм взаємодії?

- Діаграма послідовностей (Sequence Diagram);
- Діаграма комунікації (Communication Diagram);
- Діаграма часових обмежень (Timing Diagram);
- Діаграма взаємодії (Interaction Overview Diagram).

7. Для чого призначена діаграма послідовностей?

Діаграма послідовностей показує динамічну взаємодію об’єктів у часі. Вона описує порядок обміну повідомленнями між об’єктами або компонентами системи під час виконання певного сценарію.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

Основні елементи діаграми послідовностей:

- Актори або об’єкти (учасники процесу);
- Життєві лінії (Lifelines);
- Повідомлення (Messages) - синхронні, асинхронні, відповідні;
- Активності (Activation bars);
- Фрагменти (alt, loop, opt) - для умов і повторень.

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?

Діаграма послідовностей деталізує сценарії, які описані на діаграмі варіантів

використання. Тобто діаграма варіантів використання визначає, що робить система, а діаграма послідовностей показує як саме відбувається взаємодія крок за кроком.

10. Як діаграми послідовностей пов’язані з діаграмами класів?

Діаграма послідовностей показує динаміку взаємодії об’єктів, які є екземплярами класів, зображених на діаграмі класів. Тобто діаграма класів описує структуру системи, а діаграма послідовностей - поведінку та обмін повідомленнями між цими класами під час виконання сценаріїв.