МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

# ЛАБОРАТОРНА РОБОТА №3

ТЕМА: «Основи проектування розгортання»

Виконала:

Студентка групи ІА-34

Потейчук С.А.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

## Теоретичні відомості

Діаграма розгортання слугує для візуалізації фізичного розподілу програмних компонентів по апаратних вузлах. Вона є фінальним ланкою в ланцюжку проектування системи, показуючи, як програмні артефакти будуть розміщені на реальному обладнанні. Ключовим поняттям тут є вузол (node), який представляє будь-який обчислювальний ресурс. Вузли бувають двох типів: пристрої (device) — фізичне обладнання, таке як сервери, комп'ютери або мобільні пристрої, та середовища виконання (execution environment) — програмні платформи, що можуть виконувати інше програмне забезпечення, наприклад, операційна система, веб-браузер, віртуальна машина Java або контейнер Docker.

Між вузлами встановлюються зв'язки, які показують канали комунікації та часто специфікують використовуваний протокол (наприклад, HTTP, TCP/IP) або технологію. Безпосередньо всередині вузлів розміщуються артефакти — фізичні реалізації програмних компонентів у вигляді файлів: виконувані файли (.exe, .jar), бібліотеки (.dll), скрипти, файли конфігурації або бази даних. Існує два рівня деталізації діаграм розгортання: описові, які оперують типами вузлів та артефактів (наприклад, "Веб-сервер", "Клієнтський ПК"), і екземплярні, які описують конкретні сутності.

Діаграма компонентів зосереджена на архітектурі самої програмної системи, демонструючи її розбиття на структурні модулі — компоненти — та залежності між ними. Компонент є модульною частиною системи, що інкапсулює її вміст і має чітко визначені інтерфейси для взаємодії з зовнішнім середовищем. Інтерфейси реалізуються через порти, до яких приєднуються конектори.

Існують різні погляди на компонентне моделювання.

Логічне розбиття. Система представляється як набір автономних, функціонально завершених модулів, що взаємодіють через чітко визначені інтерфейси. Це допомагає проєктувати архітектуру системи.

Фізичне розбиття. Діаграма відображає реальні файли реалізації (наприклад, .exe, .dll, .jar) та компіляційні залежності між ними. Така модель незамінна для планування процесів збірки та розгортання, оскільки показує, які саме компоненти повинні бути включені в інсталяційний пакет, і як зміни в одному компоненті впливають на інші.

Основні цілі створення діаграми компонентів включають візуалізацію архітектури на рівні модулів, проєктування та документування інтерфейсів, забезпечення багаторазового використання компонентів і планування процесів збірки та розгортання.

Діаграма послідовностей належить до діаграм поведінки і детально моделює динаміку взаємодії об'єктів у часовій послідовності. Вона фокусується на порядку передачі повідомлень між об'єктами в межах конкретного сценарію використання системи. Основний каркас діаграми формують об'єкти або актори, розміщені вгорі, та їхні лінії життя — вертикальні пунктирні лінії, що відображають існування об'єкта в часі. Взаємодія відбувається через повідомлення, які передаються між лініями життя. Розрізняють різні типи повідомлень: синхронні, асинхронні (відкрита стрілка, не чекають відповіді) та повернення (пунктирна стрілка).

Періоди активної роботи об'єкта позначаються прямокутниками активності на його лінії життя. Для моделювання складних сценаріїв використовуються фрагменти взаємодії, такі як opt (необов'язкова послідовність), alt (альтернативні варіанти, аналог if-else), loop (циклічне виконання) та par (паралельне виконання). Діаграми послідовностей є критично важливими інструментами для аналізу та проектування логіки взаємодії, виявлення помилок проектування, документування складних алгоритмів та планування тестування системної інтеграції.

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 лабораторній роботі.

## Хід роботи

7. Редактор зображень

Редактор зображень має такі функціональні можливості: відкриття/збереження зображень у найпопулярніших форматах (5 на вибір студента), застосування ефектів, наприклад поворот, розтягування, стиснення, кадрування зображення, можливість створення колажів шляхом «нашарування» зображень.

З попередньої лабораторної роботи отримали діаграму варіантів використання та діаграму класів:

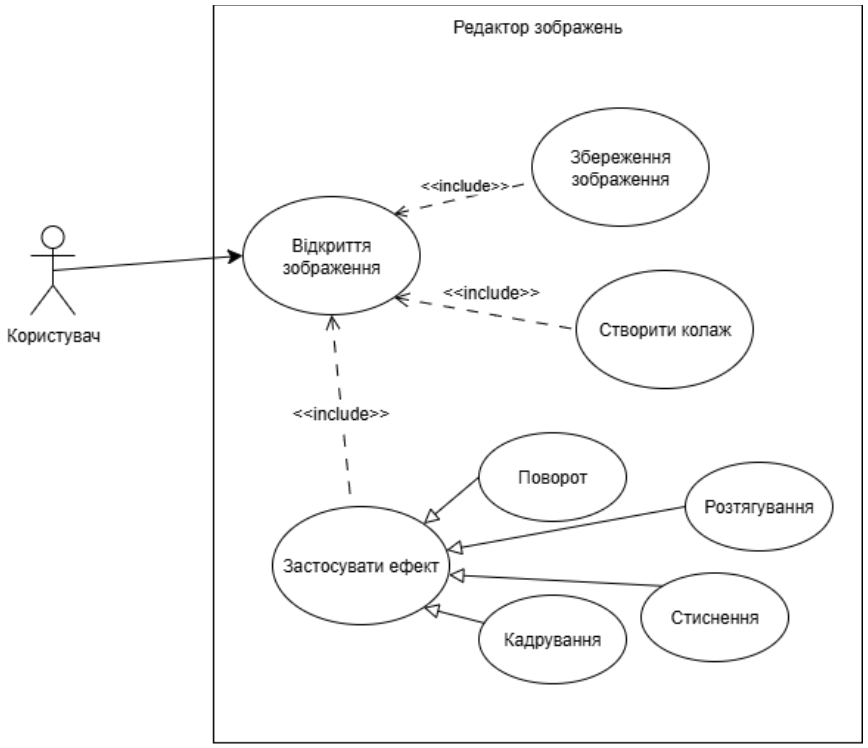


Рисунок 1 - Діаграма варіантів використання

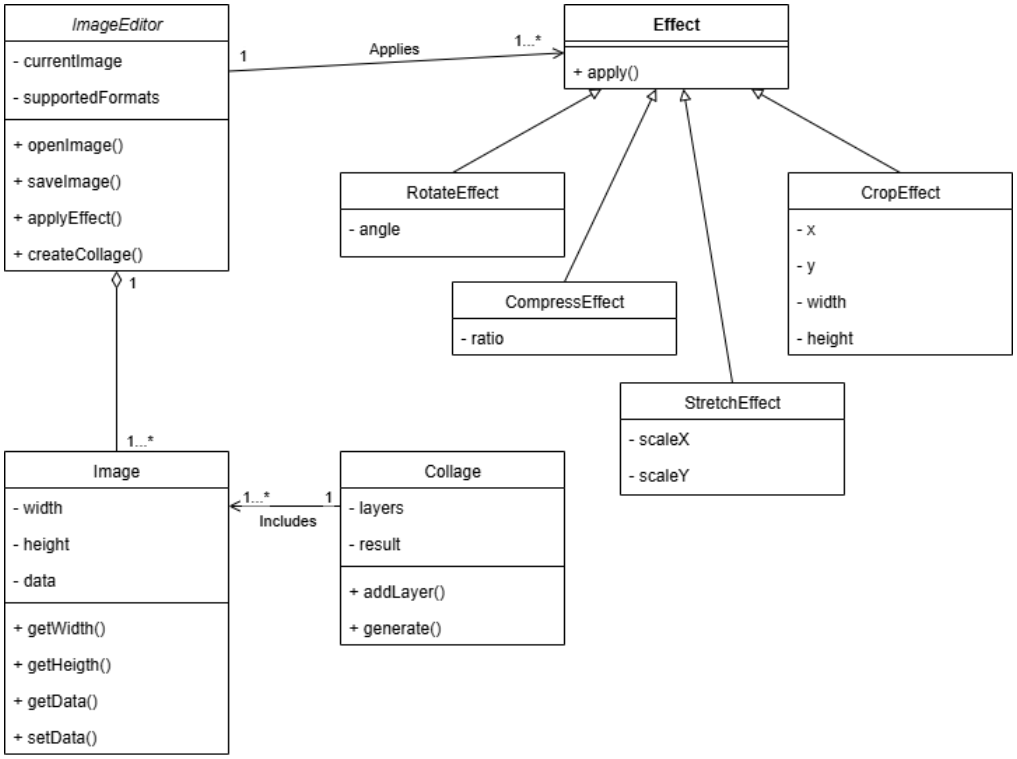


Рисунок 2 - Діаграма класів системи

Зв’язки:

- ImageEditor агрегує Image (поточне зображення).
- ImageEditor застосовує Effect (асоціація).
- Effect – узагальнюючий клас; RotateEffect, StretchEffect, CompressEffect, CropEffect – наслідують його.
- Collage містить багато Image (асоціація).

Проаналізувавши ці діаграми, проєктуємо діаграму розгортання використання відповідно до обраної теми лабораторного циклу:

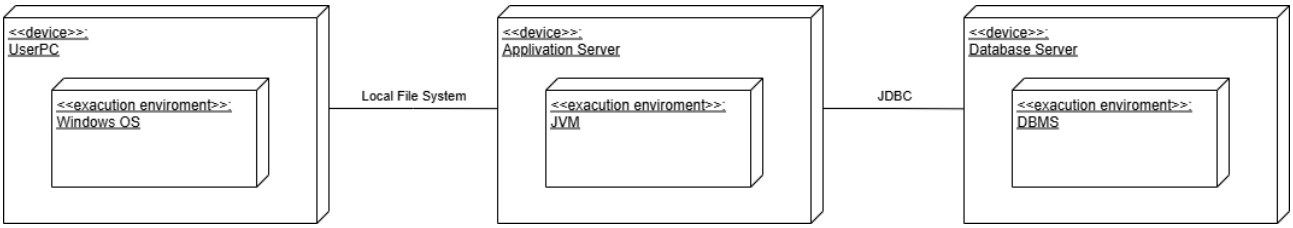


Рисунок 3 – Діаграма розгортання

<<device>> UserPC — фізичний пристрій користувача.

<<execution environment>> Windows OS — операційна система, у якій працює застосунок.

Усередині — десктопний редактор зображень.

<<device>> Application Server — серверна машина (локальна).

<<execution environment>> JVM — середовище виконання бекенду.

Всередині — сервіси обробки зображень та ефектів.

<<device>> Database Server — окремий сервер для зберігання даних.

<<execution environment>> DBMS — система керування базами даних.

Зберігає таблиці Images, Collages, CollageImages, Effects.

Розробляємо діаграму компонентів для проєктованої системи:

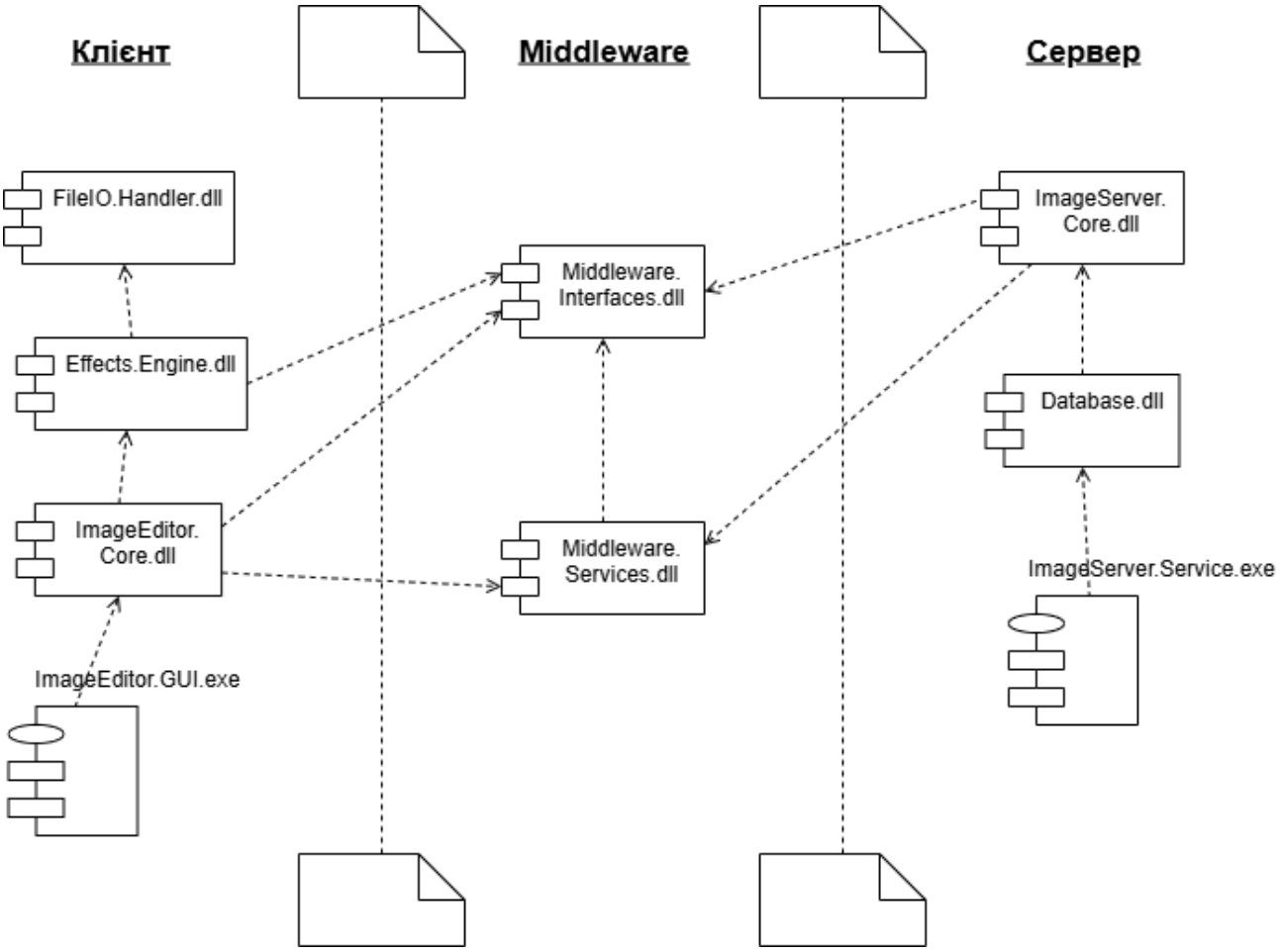


Рисунок 4 – Діаграма компонентів

Клієнт:

- ImageEditor.GUI.exe – виконуваний файл користувацького інтерфейсу.
- ImageEditor.Core.dll – головна логіка редактора (керування зображеннями, виклики ефектів).
- Effects.Engine.dll – бібліотека обробки зображень.
- FileIO.Handler.dll – робота з форматами файлів.

Middleware:

- Middleware.Interfaces.dll – спільні інтерфейси та DTO, що забезпечують узгодженість між клієнтом і сервером.
- Middleware.Services.dll – бізнес-логіка, що може бути спільною для обох сторін.

Сервер:

- ImageServer.Service.exe – серверний застосунок, що обробляє запити.
- ImageServer.Core.dll – серверна бізнес-логіка.
- Database.dll – модуль доступу до бази даних.

Розробляємо дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі:

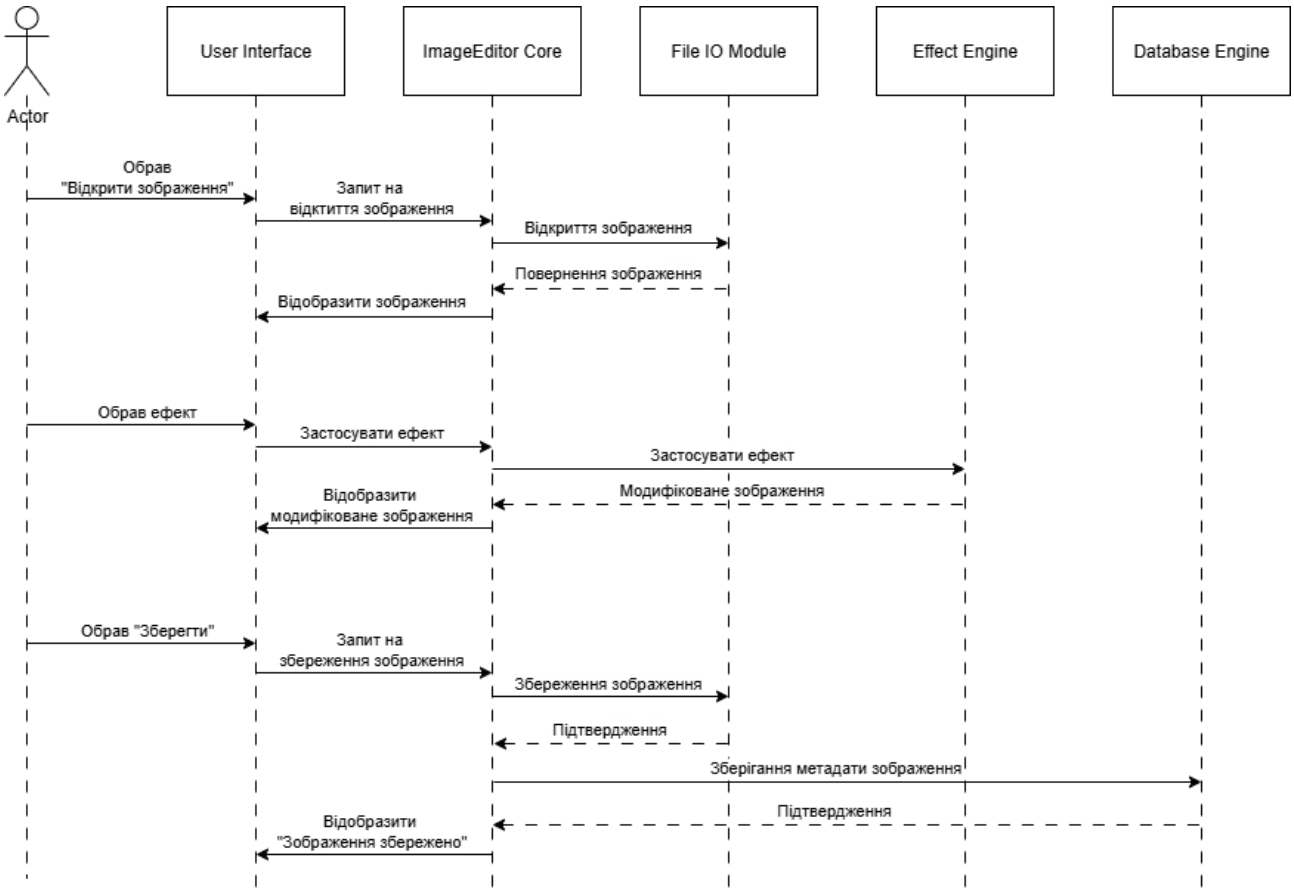


Рисунок 5 – Діаграма послідовностей для відкриття зображення, застосування ефекту та зберігання результату.

Опис сценарію:

1. Користувач натискає «Відкрити файл».
2. Інтерфейс викликає ядро програми для завантаження зображення.
3. Ядро звертається до модуля File I/O, який відкриває файл.
4. Користувач застосовує ефект (наприклад, обертання).
5. Інтерфейс викликає ядро, яке передає зображення у модуль Effect Engine.

6. Effect Engine модифікує зображення та повертає його ядру.
7. Користувач натискає «Зберегти».
8. Ядро викликає File I/O для збереження результату.
9. Інформація про зображення записується в базу даних.

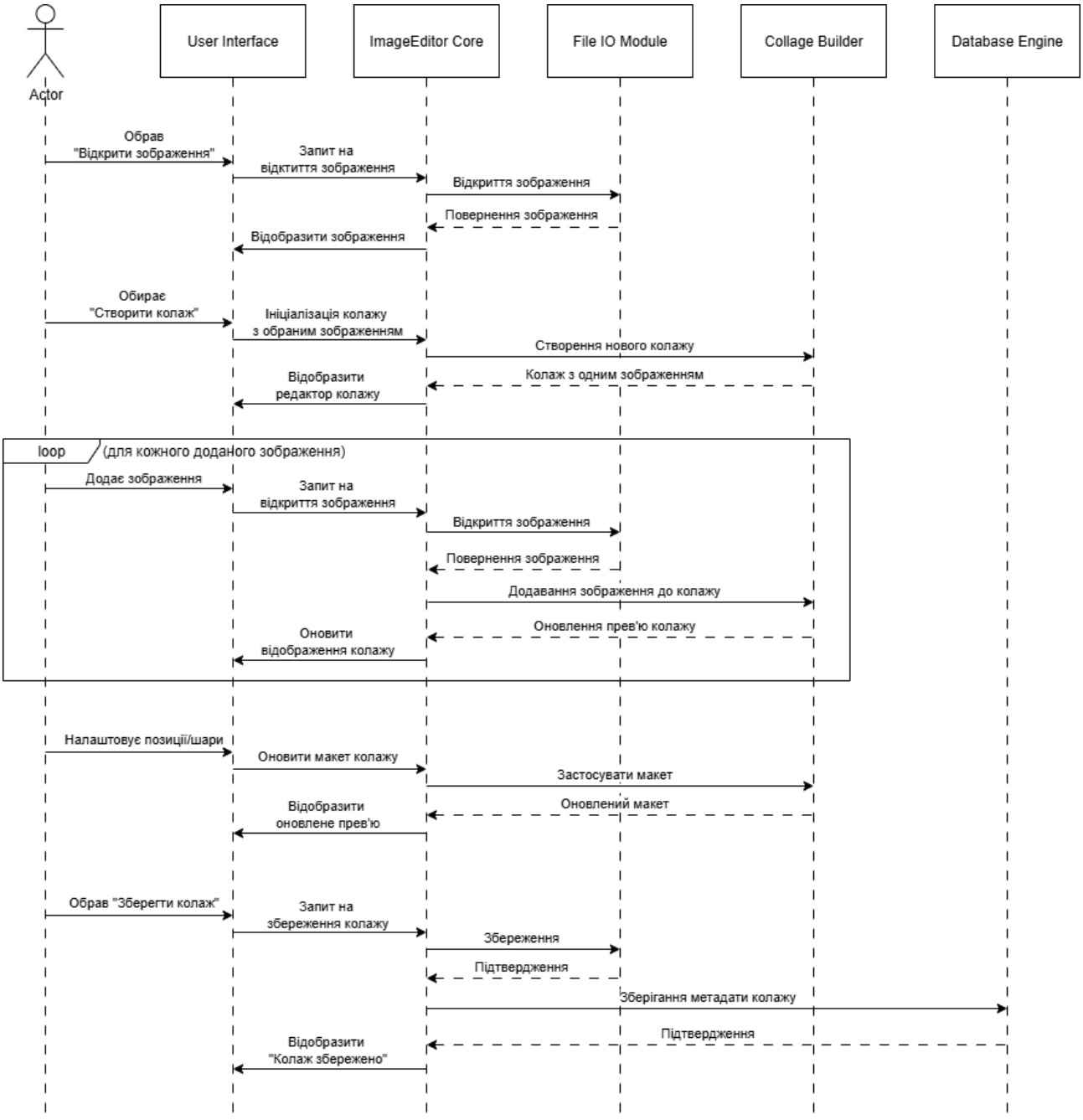


Рисунок 6 – Діаграма послідовностей для створення колажу.

Опис сценарію:

1. Користувач натискає «Відкрити файл».
2. Інтерфейс викликає ядро програми для завантаження зображення.

3. Ядро звертається до модуля File I/O, який відкриває файл.
4. Користувач натискає «Створити колаж».
5. Інтерфейс запитує ядро редактора ініціалізувати колаж.
6. Користувач послідовно додає зображення (відкриття через File I/O).
7. Кожне зображення передається у модуль Collage Builder.
8. Користувач налаштовує позиції та порядок шарів.
9. Collage Builder формує готовий колаж і повертає його ядру.
10. Користувач натискає «Зберегти колаж».
11. Колаж зберігається через File I/O, а інформація — у базу даних.

Код системи, який було додано в цій лабораторній роботі.

Лістинг 1 – JpaUtil.java

```
package util;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.Persistence;

public class JpaUtil {
    private static final EntityManagerFactory emf;

    static {
        emf = Persistence.createEntityManagerFactory("ImageEditorPU");
    }

    public static EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

Лістинг 2 – ImageRepository.java

```
package repository;

import jakarta.persistence.EntityManager;
import model.Image;
import util.JpaUtil;
import java.util.List;

public class ImageRepository {
    public void save(Image image) {
        EntityManager em = JpaUtil.getEntityManager();
        em.getTransaction().begin();
        em.persist(image);
        em.getTransaction().commit();
        em.close();
    }
```

```
    public List<Image> findAll() {
        EntityManager em = JpaUtil.getEntityManager();
        List<Image> result = em.createQuery("SELECT i FROM Image i",
Image.class).getResultList();
        em.close();
        return result;
    }

    public Image findById(int id) {
        EntityManager em = JpaUtil.getEntityManager();
        Image img = em.find(Image.class, id);
        em.close();
        return img;
    }
}
```

## Лістинг 4 – CollageRepository.java

```
package repository;

import jakarta.persistence.EntityManager;
import model.Collage;
import util.JpaUtil;
import java.util.List;

public class CollageRepository {
    public void save(Collage collage) {
        EntityManager em = JpaUtil.getEntityManager();
        em.getTransaction().begin();
        em.persist(collage);
        em.getTransaction().commit();
        em.close();
    }

    public List<Collage> findAll() {
        EntityManager em = JpaUtil.getEntityManager();
        List<Collage> list = em.createQuery("SELECT c FROM Collage c",
Collage.class).getResultList();
        em.close();
        return list;
    }
}
```

## Лістинг 5 – EffectRepository.java

```
package repository;

import jakarta.persistence.EntityManager;
import model.Effect;
import util.JpaUtil;

public class EffectRepository {
    public void save(Effect effect) {
        EntityManager em = JpaUtil.getEntityManager();
        em.getTransaction().begin();
        em.persist(effect);
        em.getTransaction().commit();
        em.close();
    }
}
```

Лістинг 6 – ImageService.java

```
package service;

import model.Image;
import repository.ImageRepository;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.List;

public class ImageService {
    private final ImageRepository repo;

    public ImageService(ImageRepository repo) {
        this.repo = repo;
    }

    public Image openImage(File file) throws IOException {
        byte[] data = Files.readAllBytes(file.toPath());
        Image img = new Image();
        img.setFileName(file.getName());
        img.setFormat("PNG");
        img.setData(data);
        repo.save(img);
        return img;
    }

    public List<Image> getAll() {
        return repo.findAll();
    }
}
```

Лістинг 7 – CollageService.java

```
package service;

import model.Collage;
import repository.CollageRepository;
import java.util.List;

public class CollageService {
    private final CollageRepository repo;

    public CollageService(CollageRepository repo) {
        this.repo = repo;
    }

    public void save(Collage collage) {
        repo.save(collage);
    }

    public List<Collage> getAll() {
        return repo.findAll();
    }
}
```

Лістинг 8 – EffectService.java

```
package service;

import model.Effect;
import repository.EffectRepository;

public class EffectService {
    private final EffectRepository repo;

    public EffectService(EffectRepository repo) {
        this.repo = repo;
    }

    public void applyEffect(Effect effect) {
        repo.save(effect);
    }
}
```

## Лістинг 9 – MainForm.java

```
package ui;

import service.ImageService;
import javax.swing.*;
import java.awt.*;
import java.io.File;

public class MainForm extends JFrame {
    private final ImageService imageService;
    private JLabel imageLabel = new JLabel();

    public MainForm(ImageService imageService) {
        super("Image Editor");
        this.imageService = imageService;

        JButton openBtn = new JButton("Відкрити зображення");
        JButton collageBtn = new JButton("Створити колаж");

        openBtn.addActionListener(e -> openImage());
        collageBtn.addActionListener(e -> openCollageEditor());

        JPanel topPanel = new JPanel();
        topPanel.add(openBtn);
        topPanel.add(collageBtn);

        add(topPanel, BorderLayout.NORTH);
        add(new JScrollPane(imageLabel), BorderLayout.CENTER);

        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    private void openImage() {
        JFileChooser chooser = new JFileChooser();
        if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            File file = chooser.getSelectedFile();
            try {
                var img = imageService.openImage(file);
                ImageIcon icon = new ImageIcon(img.getData());
                imageLabel.setIcon(icon);
            } catch (Exception ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(this, "Помилка при відкритті
файлу");
```

```
            }
        }
    }

    private void openCollageEditor() {
        new CollageEditorForm().setVisible(true);
    }
}
```

## Лістинг 10 – CollageEditorForm.java

```
package ui;

import service.ImageService;
import javax.swing.*;
import java.awt.*;
import java.io.File;

public class MainForm extends JFrame {
    private final ImageService imageService;
    private JLabel imageLabel = new JLabel();

    public MainForm(ImageService imageService) {
        super("Image Editor");
        this.imageService = imageService;

        JButton openBtn = new JButton("Відкрити зображення");
        JButton collageBtn = new JButton("Створити колаж");

        openBtn.addActionListener(e -> openImage());
        collageBtn.addActionListener(e -> openCollageEditor());

        JPanel topPanel = new JPanel();
        topPanel.add(openBtn);
        topPanel.add(collageBtn);

        add(topPanel, BorderLayout.NORTH);
        add(new JScrollPane(imageLabel), BorderLayout.CENTER);

        setSize(600, 400);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    private void openImage() {
        JFileChooser chooser = new JFileChooser();
        if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION) {
            File file = chooser.getSelectedFile();
            try {
                var img = imageService.openImage(file);
                ImageIcon icon = new ImageIcon(img.getData());
                imageLabel.setIcon(icon);
            } catch (Exception ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog(this, "Помилка при відкритті
файлу");
            }
        }
    }

    private void openCollageEditor() {
        new CollageEditorForm().setVisible(true);
    }
}
```

Лістинг 11 – App.java

```
import repository.*;
import service.*;
import ui.MainForm;

public class App {
    public static void main(String[] args) {
        ImageRepository imgRepo = new ImageRepository();
        CollageRepository colRepo = new CollageRepository();
        EffectRepository effRepo = new EffectRepository();

        ImageService imgService = new ImageService(imgRepo);
        CollageService colService = new CollageService(colRepo);
        EffectService effService = new EffectService(effRepo);

        javax.swing.SwingUtilities.invokeLater(() -> {
            MainForm form = new MainForm(imgService);
            form.setVisible(true);
        });
    }
}
```

# Висновки

У ході виконання лабораторної роботи було спроєктовано та реалізовано десктопний застосунок «Редактор зображень», який підтримує основні функціональні можливості: відкриття та збереження зображень у популярних форматах, застосування різних графічних ефектів (поворот, розтягування, стиснення, кадрування), а також створення колажів шляхом нашарування декількох зображень.

На етапі проєктування були розроблені основні UML-діаграми, які відображають структуру та поведінку системи. Діаграма розгортання визначила апаратне та програмне середовище виконання застосунку. Діаграма компонентів дозволила структурувати систему на логічні модулі та встановити залежності між ними. Діаграми послідовностей описали динаміку роботи системи в різних сценаріях (відкриття зображення, створення колажу тощо).

Завдяки розробленим UML-діаграмам проєктування стало більш структурованим і зрозумілим, а реалізація – гнучкою та масштабованою. У результаті було здобуто практичні навички побудови UML-моделей та організації багаторівневої архітектури застосунків.