МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ЛАБОРАТОРНА РОБОТА №2

ТЕМА: «Основи проектування»

Виконала:

Студентка групи ІА-34

Потейчук С.А.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Теоретичні відомості

UML — це універсальна графічна мова, яка дає змогу описувати, проєктувати та документувати програмне забезпечення, бізнес-процеси та інші складні системи. Вона поєднує напрацювання багатьох методологій програмної інженерії, тому придатна і для концептуального, і для логічного, і для фізичного моделювання. У межах об’єктно-орієнтованого аналізу та проєктування UML підтримує послідовний перехід від загальних моделей до конкретних, поступово додаючи деталі й відображаючи різні аспекти майбутньої системи. При цьому модель можна розглядати на кількох рівнях абстракції — від концептуального (початкового, найзагальнішого) до логічного (структура, взаємодії) й фізичного (реальні компоненти та середовище).

Усі ці уявлення фіксуються через діаграми. Діаграма — це графічне подання елементів моделі й зв’язків між ними у вигляді графа зі спеціальною семантикою для вузлів і ребер. У нотації UML визначено кілька основних типів діаграм: варіантів використання (use case), класів, кооперації, послідовності, станів, діяльності, компонентів і розгортання. Разом вони дають повну картину функціонування системи: від її зовнішніх вимог та сценаріїв роботи до внутрішньої структури й фізичного розміщення компонентів.

Діаграма варіантів використання — це вихідна концептуальна модель, яка показує межі функціональності системи, її основні сценарії та вимоги. Вона створюється на етапі збору й аналізу вимог, коли бізнес-аналітики й розробники уточнюють, що саме система повинна робити. Ця діаграма не описує внутрішню будову, а лише відображає взаємодії між зовнішніми користувачами або системами (акторами) і самою системою.

Актори — це будь-які зовнішні суб’єкти: люди, пристрої, інші програми або системи, які взаємодіють із моделлю. Їх зазвичай розглядають як ролі: наприклад, «покупець», «касир», «адміністратор». Варіанти використання (use cases) описують послуги чи сценарії, які система надає актору, тобто послідовності дій, що виконуються системою під час діалогу з актором

(реєстрація, авторизація, оформлення замовлення, перевірка рахунку тощо). Вони позначаються еліпсами з короткими інформативними назвами.

Зв’язки між акторами й варіантами використання фіксують тип взаємодії.

Асоціація — найзагальніший зв’язок між актором і варіантом використання, показує, що актор користується певною функцією системи. Вона може бути ненаправленою (коли напрямок неважливий або ще не визначено) чи спрямованою (коли видно, хто ініціює взаємодію).

Узагальнення — спадкування атрибутів чи поведінки між елементами одного типу. Наприклад, актор «Адміністратор» успадковує атрибути «Користувача» й має додаткові. Аналогічно для варіантів використання: «Оплата замовлення банківською карткою» є спеціалізацією «Оплата замовлення».

Залежність — відображає, що зміни одного елемента можуть впливати на інший. Спеціальні випадки залежності:

«Включення» (include) — коли поведінка одного варіанта використання завжди містить поведінку іншого (наприклад, оформлення замовлення обов’язково включає автентифікацію користувача).

«Розширення» (extend) — коли базовий сценарій може бути доповнений додатковими діями, які вставляються за певних умов.

Разом такі діаграми й зв’язки створюють цілісну картину системи на концептуальному рівні, даючи базу для побудови діаграм класів, послідовностей, компонентів тощо. Це дозволяє ще до проєктування та реалізації отримати повне й узгоджене уявлення про функціональність, поведінку та структуру складної системи.

Тема: Основи проектування.

Мета: Обрати зручну систему побудови UML-діаграм та навчитися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

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.

Проаналізувавши тему, проєктуємо діаграму варіантів використання відповідно до обраної теми лабораторного циклу:

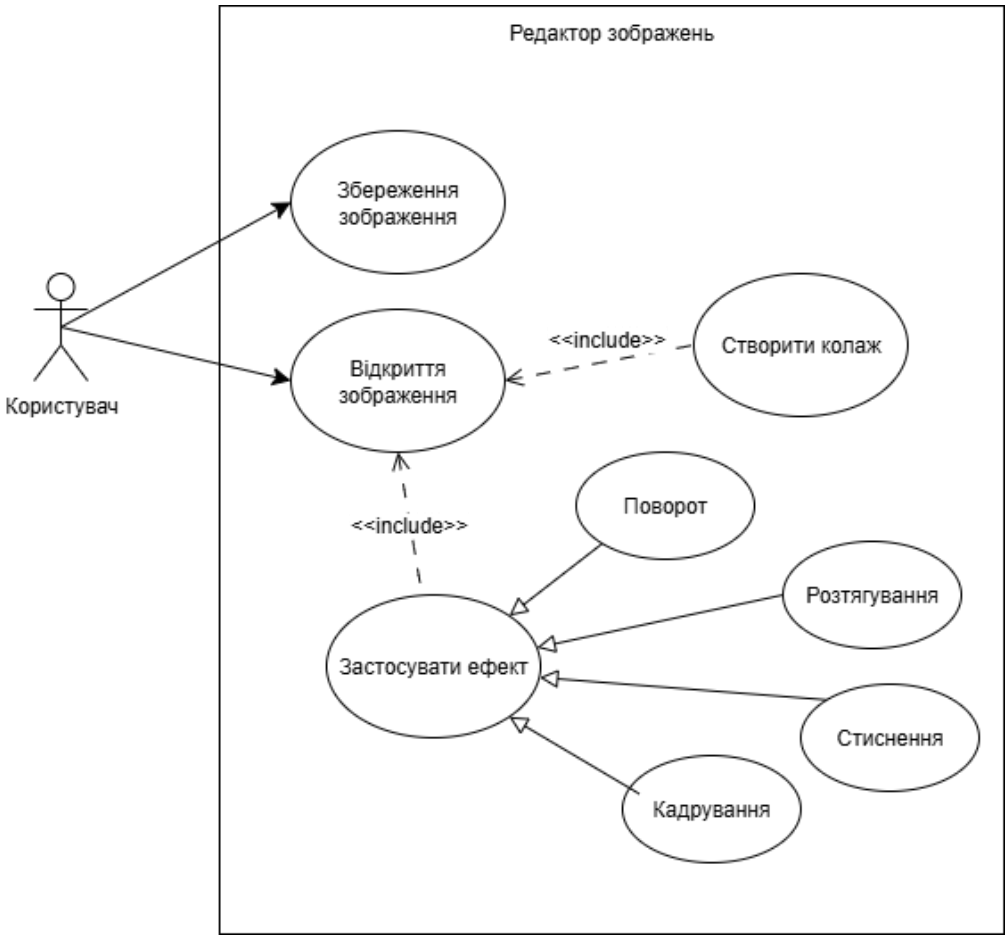

Рисунок 1 - Діаграма варіантів використання

Проєктуємо діаграму класів предметної області:

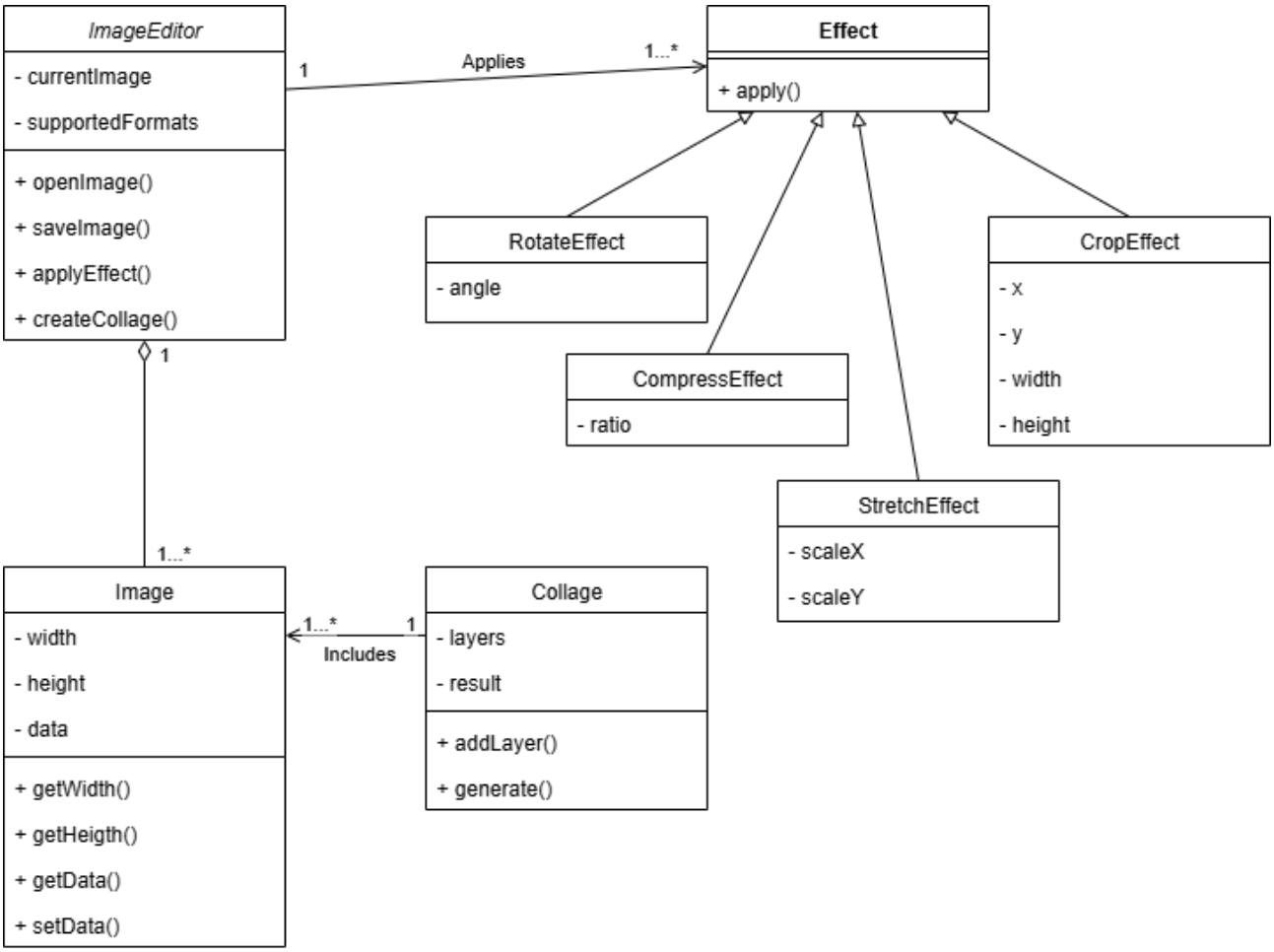

Рисунок 2 - Діаграма класів системи

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).

Вибираємо 3 варіанти використання та напишемо за ними сценарії використання:

Варіант використання 1: «Відкрити та редагувати зображення»

Передумови: Немає.

Постумови:

- У разі успішного виконання користувач отримує у вікні редактора завантажене зображення й може застосовувати до нього ефекти.
- У протилежному випадку стан системи не змінюється, зображення не завантажено.

Взаємодіючі сторони: Користувач, Редактор зображень.

Короткий опис: Цей варіант використання визначає процес відкриття користувачем зображення у редакторі та застосування ефектів.

Основний потік подій.

Цей варіант використання починає виконуватися, коли користувач хоче відкрити файл для редагування.

1. Система пропонує вибрати файл із диска.
2. Користувач обирає зображення у підтримуваному форматі (наприклад, JPEG, PNG, BMP, TIFF, GIF).
3. Система завантажує вибране зображення у робоче поле редактора.
4. Користувач застосовує необхідні ефекти: поворот, розтягування, стиснення, кадрування або створює колаж (нашарування).
5. Система відображає результат застосованих змін у вікні попереднього перегляду.
6. Користувач зберігає відредаговане зображення у вибраному форматі.

Винятки

Виняток №1: Непідтримуваний формат файлу. Якщо користувач обирає файл у форматі, що не підтримується, система виводить повідомлення про

помилку. Користувач може повернутися до вибору файлу або відмовитись від операції, при цьому виконання варіанта використання завершується.

Виняток №2: Невдале збереження. Якщо під час збереження виникла помилка, система виводить повідомлення про помилку. Користувач може вибрати інше місце/назву файлу або скасувати збереження.

Примітки: Відсутні.

Варіант використання 2: «Поворот зображення»

Передумови: У редакторі вже відкрито зображення.

Постумови:

- У разі успішного виконання зображення повернене на вибраний кут.
- У протилежному випадку стан зображення не змінюється.

Взаємодіючі сторони: Користувач, Редактор зображень.

Короткий опис: Цей варіант використання визначає процес повороту завантаженого у редактор зображення.

Основний потік подій.

- Користувач відкриває вже завантажене зображення у редакторі.
- Система пропонує інструмент «Поворот».
- Користувач вибирає кут або напрямок повороту.
- Система виконує поворот і відображає оновлене зображення у вікні редактора.

Винятки

Виняток №1: Немає завантаженого зображення. Якщо користувач запускає інструмент «Поворот», коли зображення не відкрито, система виводить повідомлення про помилку й пропонує спершу відкрити файл.

Примітки: Відсутні.

Варіант використання 3: «Створення колажу»

Передумови: Редактор зображень запущено.

Постумови:

- У разі успішного виконання створено новий колаж із кількох зображень.
- У протилежному випадку стан системи не змінюється, колаж не створено.

Взаємодіючі сторони: Користувач, Редактор зображень.

Короткий опис: Цей варіант використання визначає процес створення колажу шляхом «нашарування» кількох зображень.

Основний потік подій.

1. Користувач вибирає у меню редактора «Створити колаж».
2. Система пропонує додати зображення (відкрити файли).
3. Користувач обирає кілька зображень, які потрібно використати у колажі.
4. Система завантажує обрані зображення на робочу область колажу.
5. Користувач розміщує та масштабує зображення за власним бажанням.
6. Система відображає результат створеного колажу у вікні редактора.
7. Користувач може зберегти колаж у вибраному форматі.

Винятки

Виняток №1: Непідтримуваний формат файлу. Система виводить повідомлення про помилку й пропонує обрати інші файли.

Виняток №2: Додано занадто багато зображень. Система може обмежити кількість зображень і вивести відповідне повідомлення.

Примітки: Відсутні.

Вихідні коди класів системи

Лістинг 1 – Клас Collage

```
@Entity
@Table(name = "collages")
public class Collage {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(nullable = false)
    private String title;

    @OneToMany(mappedBy = "collage", cascade = CascadeType.ALL, orphanRemoval =
true)
    private List<CollageImage> layers = new ArrayList<>();

    public Collage() {}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public List<CollageImage> getLayers() { return layers; }
    public void setLayers(List<CollageImage> layers) { this.layers = layers; }

    public void addLayer(CollageImage layer) {
        layers.add(layer);
        layer.setCollage(this);
    }

    public void removeLayer(CollageImage layer) {
        layers.remove(layer);
        layer.setCollage(null);
    }
}
```

Лістинг 2 – Клас CollageImage

```
@Entity
@Table(name = "collage_images")
public class CollageImage {

    @EmbeddedId
    private CollageImageId id = new CollageImageId();

    @ManyToOne
    @MapsId("collageId")
    @JoinColumn(name = "collage_id")
    private Collage collage;

    @ManyToOne
    @MapsId("imageId")
    @JoinColumn(name = "image_id")
    private Image image;

    @Column(name = "layer_order")
    private int layerOrder;
```

```
    public CollageImage() {}

    public CollageImageId getId() { return id; }
    public void setId(CollageImageId id) { this.id = id; }

    public Collage getCollage() { return collage; }
    public void setCollage(Collage collage) { this.collage = collage; }

    public Image getImage() { return image; }
    public void setImage(Image image) { this.image = image; }

    public int getLayerOrder() { return layerOrder; }
    public void setLayerOrder(int layerOrder) { this.layerOrder = layerOrder; }
}
```

Лістинг 3 – Клас CompressEffect

```
@Entity
@Table(name = "compress_effects")
public class CompressEffect extends Effect {

    // Наприклад, коефіцієнт стиснення (0.0 – 1.0)
    private float compressionRatio;

    public CompressEffect() {}

    public float getCompressionRatio() {
        return compressionRatio;
    }

    public void setCompressionRatio(float compressionRatio) {
        this.compressionRatio = compressionRatio;
    }
}
```

Лістинг 4 – Клас CropEffect

```
@Entity
@Table(name = "crop_effects")
public class CropEffect extends Effect {

    private int x;
    private int y;
    private int width;
    private int height;

    public CropEffect() {}

    public int getX() { return x; }
    public void setX(int x) { this.x = x; }

    public int getY() { return y; }
    public void setY(int y) { this.y = y; }

    public int getWidth() { return width; }
    public void setWidth(int width) { this.width = width; }

    public int getHeight() { return height; }
    public void setHeight(int height) { this.height = height; }
}
```

Лістинг 5 – Клас Effect

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@Table(name = "effects")
public abstract class Effect {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(nullable = false)
    private String name;

    public Effect() {}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

Лістинг 6 – Клас Image

```
@Entity
@Table(name = "images")
public class Image {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "file_name", nullable = false)
    private String fileName;

    @Column(nullable = false)
    private String format; // PNG, JPEG...

    private int width;
    private int height;

    @Lob
    @Column(nullable = false)
    private byte[] data;

    public Image() {}

    // getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getFileName() { return fileName; }
    public void setFileName(String fileName) { this.fileName = fileName; }

    public String getFormat() { return format; }
    public void setFormat(String format) { this.format = format; }

    public int getWidth() { return width; }
    public void setWidth(int width) { this.width = width; }
```

```
    public int getHeight() { return height; }
    public void setHeight(int height) { this.height = height; }

    public byte[] getData() { return data; }
    public void setData(byte[] data) { this.data = data; }
}
```

Лістинг 7 – Клас RotateEffect

```
@Entity
@Table(name = "rotate_effects")
public class RotateEffect extends Effect {

    private float angle;

    public RotateEffect() {}

    public float getAngle() { return angle; }
    public void setAngle(float angle) { this.angle = angle; }
}
```

Лістинг 8 – Клас StretchEffect

```
@Entity
@Table(name = "stretch_effects")
public class StretchEffect extends Effect {

    // Напрямки розтягування: по ширині та по висоті
    private float horizontalScale;
    private float verticalScale;

    public StretchEffect() {}

    public float getHorizontalScale() {
        return horizontalScale;
    }

    public void setHorizontalScale(float horizontalScale) {
        this.horizontalScale = horizontalScale;
    }

    public float getVerticalScale() {
        return verticalScale;
    }

    public void setVerticalScale(float verticalScale) {
        this.verticalScale = verticalScale;
    }
}
```

Лістинг 9 – Клас models.sql

```
CREATE TABLE Images (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    FileName TEXT NOT NULL,
    Format TEXT NOT NULL,
    Width INT,
```

```
    Height INT,
    Data BLOB NOT NULL
);

CREATE TABLE Effects (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    Parameters TEXT
);

CREATE TABLE Collages (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Title TEXT NOT NULL
);

CREATE TABLE CollageImages (
    CollageId INT,
    ImageId INT,
    LayerOrder INT,
    FOREIGN KEY (CollageId) REFERENCES Collages(Id),
    FOREIGN KEY (ImageId) REFERENCES Images(Id),
    PRIMARY KEY (CollageId, ImageId)
);
```

Структура бази даних

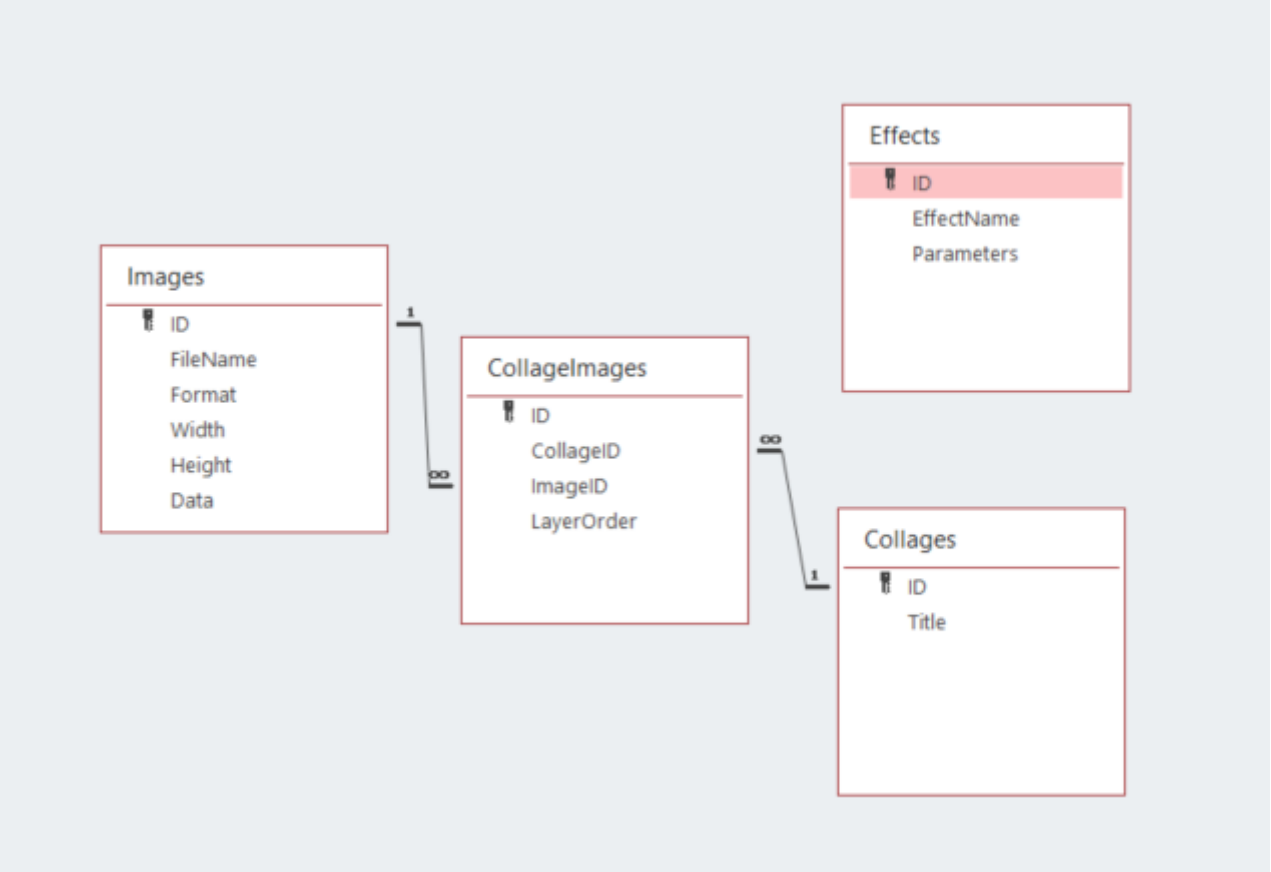

Рисунок 3 – Зображення структури бази даних

Висновки

У ході виконання лабораторної роботи було проаналізовано предметну область «Редактор зображень» і визначено основні функціональні можливості системи (відкриття та збереження зображень, застосування ефектів, створення колажів), спроєктовано UML-діаграму класів, яка відображає головні сутності системи, їх атрибути, методи та взаємозв’язки. На основі діаграми класів розроблено структуру бази даних, що включає таблиці Images, Effects, Collages та CollageImages, які забезпечують збереження і взаємозв’язки даних. Було отримано практичні навички проектування об’єктно-орієнтованої моделі системи та побудови схеми бази даних на її основі.

Таким чином, у результаті виконання роботи сформовано цілісне уявлення про процес переходу від вимог до системи до її реалізації: від варіантів використання — через діаграму класів — до структури бази даних.