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 №2

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування»

Тема роботи: 7. Редактор зображень

Виконав
студент групи ІА–33
Марченко Вадим Олександрович

Київ 2025

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.

Зміст

Короткі теоретичні відомості

Мова UML є загальноцільовою мовою візуального моделювання, яка розроблена для специфікації, візуалізації, проєктування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. В рамках мови UML уявлення про модель складної системи фіксуються у вигляді спеціальних графічних конструкцій, що отримали назву діаграм.

Діаграма (diagram) – графічне уявлення сукупності елементів моделі у формі зв'язкового графа, вершинам і ребрам (дугам) якого приписується певна семантика. Нотація канонічних діаграм є основним засобом розробки моделей мовою UML.

У нотації мови UML визначено такі види діаграм: варіантів використання (use case diagram); класів (class diagram); кооперації (collaboration diagram); послідовності (sequence diagram); станів (statechart diagram); діяльності (activity diagram); компонентів (component diagram); розгортання (deployment diagram).

Діаграма варіантів використання (Use-Cases Diagram) – це UML діаграма за допомогою якої у графічному вигляді можна зобразити вимоги до системи, що розробляється.

Діаграми варіантів використання призначені для: визначення загальної межі функціональності проєктованої системи; формулювання загальних вимоги до функціональної поведінки проєктованої системи; подальшої розробка вихідної концептуальної моделі системи (діаграми класів); створення основи для виконання аналізу, проєктування, розробки та тестування.

Діаграма варіантів використання складається з низки елементів. Основними елементами є: варіанти використання або прецеденти (use case), актор або дійова особа (actor) та відносини між акторами та варіантами використання (relationship).

Актором називається будь-який об'єкт, суб'єкт чи система, що взаємодіє з модельованою бізнес-системою ззовні для досягнення своїх цілей або вирішення певних завдань. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка служить джерелом впливу на систему, що моделюється.

Варіант використання служить для опису служб, які система надає актору. Інакше кажучи кожен варіант використання визначає набір дій, здійснюваний системою під час діалогу з актором. Кожен варіант використання являє собою послідовність дій, який повинен бути виконаний системою, що проєктується при взаємодії її з відповідним актором, самі ці дії не відображаються на діаграмі. Варіант використання відображається еліпсом, всередині якого міститься його коротке ім'я з великої літери у формі іменника або дієслова.

Відношення (relationship) – семантичний зв'язок між окремими елементами моделі.

Існують такі відносини: асоціації, узагальнення, залежність (складається з включення та розширення).

Асоціація (association) – узагальнене, невідоме ставлення між актором та варіантом використання. Позначається суцільною лінією між актором та варіантом використання. Розрізняють ненаправлену (двонаправлену) асоціацію та однонаправлену асоціацію. Ненаправлена асоціація показує взаємодію без акцента на напрямок, або коли напрямок ще не аналізувався.

Спрямована, або направлена асоціація (directed association) – також показує що актор асоціюється з варіантом використання але показує, що варіант використання ініціалізується актором. Позначається стрілкою.

Спрямована асоціація дозволяє запровадити поняття основного актора (він є ініціатором асоціації) та другорядного актора (варіант використання є ініціатором, тобто передає акторові довідкові відомості або звіт про виконану роботу).

Відношення узагальнення (generalization) – показує, що нащадок успадковує атрибути у свого прямого батьківського елементу. Тобто, один елемент моделі є спеціальним або окремим випадком іншого елемента моделі. Може застосовуватися як до акторів, так і до варіантів використання.

Графічно відношення узагальнення позначається суцільною лінією зі стрілкою у формі незафарбованого трикутника, яка вказує на батьківський варіант використання.

Відношення залежності (dependency) визначається як форма взаємозв'язку між двома елементами моделі, призначена для специфікації тієї обставини, що зміна одного елемента моделі призводить до зміни деякого іншого елемента.

Відношення включення (include) – окремий випадок загального відношення залежності між двома варіантами використання, при якому деякий варіант використання містить поведінку, визначену в іншому варіанті використання

Відношення розширення (extend) – показує, що варіант використання розширює базову послідовність дій та вставляє власну послідовність. У цьому на відміну типу відносин «включення» розширена послідовність може здійснюватися залежно від певних умов.

Для документації варіантів використання у вигляді певної специфікації та усунення неточностей і непорозуміння діаграм варіантів використання, як частину процесу збору та аналізу вимог складаються звані сценарії використання.

Сценарії використання описують варіанти використання природною мовою. Вони мають загального, шаблонного вигляду написання, проте рекомендується такий перелік для опису:

1. Передумови – умови, які повинні бути виконані для виконання даного варіанту використання;

2. Постумови – що виходить в результаті виконання даного варіанту використання;

3. Взаємодіючі сторони;

4. Короткий опис

5. Основний перебіг подій;

6. Винятки

7. Примітки

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру. Діаграма класів не відображає динамічної поведінки об'єктів зображених на ній класів. На діаграмах класів показуються класи, інтерфейси та відносини між ними.

Клас – це основний будівельний блок програмної системи. Це поняття є і в мовах програмування, тобто між класами UML та програмними класами є відповідність, що є основою для автоматичної генерації програмних кодів або для виконання реінжинірингу. Кожен клас має назву, атрибути та операції. Клас на діаграмі показується як прямокутник, розділений на 3 області. У верхній міститься назва класу, у середній – опис атрибутів (властивостей), у нижній – назви операцій – послуг, що надаються об'єктами цього класу.

На діаграмах класів зазвичай показуються асоціації та узагальнення.

Кожна асоціація несе інформацію про зв'язки між об'єктами усередині програмної системи. Найчастіше використовуються бінарні асоціації, які пов'язують два класи. Асоціація може мати назву, яка має виражати суть відображуваного зв'язку. Крім назви, асоціація може мати таку

характеристику як множинність. Вона показує, скільки об'єктів кожного класу може брати участь у асоціації.

Асоціація – найбільш загальний вид зв'язку між двома класами системи. Як правило, вона відображає використання одного класу іншим за допомогою певної якості або поля.

Узагальнення (успадкування) на діаграмах класів використовується, щоб показати зв'язок між класом-батьком та класом-нащадком. Воно вводиться на діаграму, коли виникає різновид будь-якого класу, і навіть у випадках, як у системі виявляються кілька класів, які мають подібну поведінку.

Розрізняють дві моделі бази даних – логічну та фізичну. Фізична модель бази даних представляє собор набір бінарних даних у вигляді файлів, структурованих та згрупованих згідно з призначенням (сегменти, екстенти та ін.), що використовується для швидкого та ефективного отримання інформації з жорсткого диска, а також для компактного зберігання та розміщення даних на жорсткому диску.

Логічна модель бази даних є структурою таблиць, уявлень, індексів та інших логічних елементів бази даних, що дозволяють власне програмування та використання бази даних.

Процес створення логічної моделі бази даних зветься проєктування бази даних (database design). Проєктування відбувається у зв'язку з опрацюванням архітектури програмної системи, оскільки база даних створюється зберігання даних, одержуваних з програмних класів.

Основним керівництвом під час проєктування таблиць є т. зв. нормальні форми баз даних.

Нормальна форма – властивість відношення в реляційній моделі даних, що характеризує його з погляду надмірності, що потенційно призводить до логічно помилкових результатів вибірки або зміни даних. Нормальна форма окреслюється сукупністю вимог, яким має задовольняти ставлення.

Змінна відношення знаходиться в першій нормальній формі (1НФ) тоді і тільки тоді, коли в будь-якому допустимому значенні відношення кожен його кортеж містить лише одне значення для кожного з атрибутів

Змінна відношення знаходиться в другій нормальній формі тоді і тільки тоді, коли вона знаходиться в першій нормальній формі, і кожен неключовий атрибут функціонально повно залежить від її потенційного ключа

Змінна відношення знаходиться у третій нормальній формі тоді і лише тоді, коли вона знаходиться у другій нормальній формі, і відсутні транзитивні функціональні залежності неключових атрибутів від ключових.

Змінна відношення знаходиться в нормальній формі Бойса-Кодда (інакше – в посиленій третій нормальній формі) тоді і тільки тоді, коли кожна її нетривіальна і неприведена зліва функціональна залежність має в якості свого детермінанта певний потенційний ключ.

Хід роботи

Аналіз вимог та проектування діаграми варіантів використання

Редактор зображень є прикладним програмним забезпеченням, яке дозволяє користувачам виконувати базові та розширені операції з графічними файлами. Основними функціональними можливостями редактора є:

- відкриття та збереження зображень у найпопулярніших форматах (наприклад, JPEG, PNG, BMP, GIF, TIFF);
- застосування ефектів і трансформацій, таких як поворот, розтягування, стиснення та кадрування;
- створення колажів на основі композиції з кількох зображень, у тому числі шляхом їхнього нашарування;
- забезпечення інтеграції з клієнт-серверною архітектурою для збереження або обміну результатами роботи.

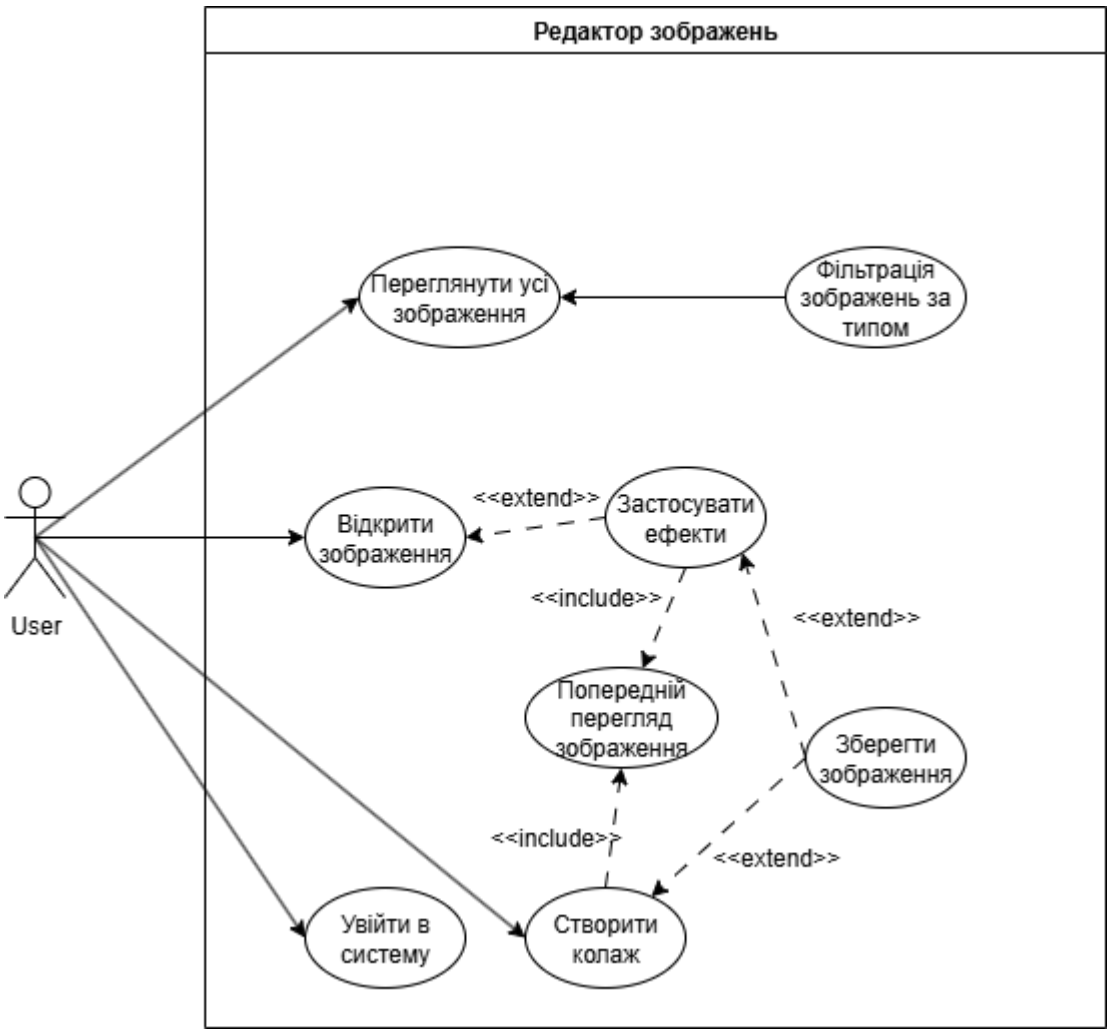


Рисунок 1. - Діаграма варіантів використання

У системі виділяється основний користувач: Зареєстрований користувач (User).

Користувач має повний доступ до функцій редактора. Він може застосовувати різноманітні ефекти до зображень (поворот, кадрування, масштабування), створювати колажі шляхом нашарування зображень, скасовувати та повторювати дії (undo/redo), переглядати зміни у реальному часі та зберігати проекти для подальшого редагування. Крім того, користувач може експортувати готові зображення у різних форматах і ділитися ними.

Сценарії використання для 3 варіантів використання

Сценарій №1: Застосування ефектів до зображення

Передумови: Користувач відкрив потрібне зображення для редагування.

Постумови:

1) Ефект успішно застосований до вибраного зображення, зміни відображаються на екрані.

2) Якщо користувач скасує дію – зображення повернеться у попередній стан.

Взаємодіючі сторони: Користувач, Веб-додаток «Редактор зображень».

Короткий опис: Користувач застосовує один із доступних ефектів до відкритого зображення. Система обробляє зображення та відображає результат у вікні попереднього перегляду.

Основний перебіг подій:

1) Користувач відкриває меню «Ефекти».

2) Система відображає список доступних ефектів.

3) Користувач обирає бажаний ефект зі списку.

4) Система застосовує вибраний ефект до зображення.

5) Користувач бачить оновлене зображення з накладеним ефектом.

6) Користувач може підтвердити зміни або скасувати дію.

Винятки:

1) Пункт 4: Якщо вибраний ефект не може бути застосований через технічну помилку – система виводить повідомлення «Помилка застосування ефекту. Спробуйте пізніше».

2) Пункт 6: Якщо користувач натискає «Скасувати» – система відновлює попередній стан зображення.

Примітки: Застосування ефекту не зберігає остаточні зміни, поки користувач не виконає явне збереження зображення.

Сценарій №2: Відкрити зображення

Передумови: Користувач знаходиться на сторінці редактора. У нього є доступ до файлу зображення на комп’ютері.

Постумови: Вибране зображення успішно завантажено у редактор. Користувач бачить його на робочій панелі та може надалі застосовувати до нього інструменти редагування.

Взаємодіючі сторони: Гість або зареєстрований користувач, Веб-додаток «Редактор зображень».

Короткий опис: Користувач відкриває зображення у редакторі для подальшого редагування. Система надає можливість вибрати файл із локального комп’ютера.

Основний перебіг подій:

1) Користувач натискає кнопку «Відкрити зображення».

2) Користувач обирає потрібний файл та підтверджує завантаження.

3) Система перевіряє формат зображення (JPG, PNG, GIF, BMP, TIFF).

4) Якщо формат підтримується, система завантажує зображення у робочу область редактора.

5) Користувач бачить зображення на екрані та може почати редагування.

Винятки:

1) Пункт 3, непідтримуваний формат: Система повідомляє користувача про помилку та пропонує вибрати інший файл.

2) Пункт 2, проблема з доступом до файлу: Система повідомляє, що файл недоступний.

Примітки: Цей варіант використання доступний як гостю, так і зареєстрованому користувачу. Завантажене зображення не зберігається автоматично, а лише використовується для поточної сесії. У випадку авторизованого користувача система може дозволяти зберегти вибране зображення як проект.

Сценарій №3: Фільтрація зображень за типом

Передумови:

1) Користувач пройшов авторизацію і працює у системі під своїм обліковим записом.

2) У користувача є завантажені або збережені зображення різних форматів.

Постумови: На екрані відображаються лише ті зображення, які відповідають обраному користувачем формату файлу.

Взаємодіючі сторони: Зареєстрований користувач, Веб-додаток «Редактор зображень».

Короткий опис: Зареєстрований користувач може відфільтрувати список своїх зображень за певним форматом файлу (наприклад, переглянути тільки PNG-файли). Це спрощує роботу з великою кількістю завантажених матеріалів.

Основний перебіг подій:

1) Користувач відкриває розділ зі своїми збереженими або завантаженими зображеннями.

2) Система відображає список усіх доступних зображень.

3) Користувач натискає кнопку «Фільтрація» та обирає потрібний формат (наприклад, PNG).

4) Система обробляє запит та відображає лише ті файли, які відповідають обраному типу.

5) Користувач переглядає відфільтровані результати і може обрати потрібне зображення для редагування.

Винятки:

1) Пункт 2, проблема із завантаженням списку: система виводить повідомлення про помилку доступу до бібліотеки зображень.

2) Пункт 3, вибрано формат, для якого немає жодного зображення: система повідомляє «Файлів даного формату не знайдено» та пропонує обрати інший тип.

Примітки: Фільтрація доступна лише авторизованим користувачам, оскільки вона вимагає доступу до особистої бібліотеки файлів. Система може підтримувати комбінаційну фільтрацію. За замовчуванням відображаються всі файли незалежно від формату.

Діаграма класів

На наступному етапі проєктування була створена модель предметної області, яка відображає основні сутності системи та зв’язки між ними. Ця модель дозволяє зрозуміти логічну структуру даних без урахування технічних деталей реалізації, таких як мова програмування чи тип бази даних.

Центральними сутностями виступають:

1) User (Користувач), що представляє будь-якого актора в системі (Зареєстрований користувач).

2) Image (Зображення), що є сутністю для зберігання інформації про зображення користувача, включаючи його назву, формат, шлях до файлу та застосовані ефекти.

3) Effect (Ефект), що описує трансформації або модифікації зображення (поворот, кадрування, стиснення тощо). Кожен ефект пов’язаний з конкретним зображенням.

4) Collage (Колаж), що дозволяє об’єднувати декілька зображень у єдину композицію. Колаж має власника та може містити кілька зображень.

Зв’язки між цими сутностями реалізують бізнес-логіку системи:

1) Кожен Користувач може мати багато Зображень та Колажів.

2) Кожне Зображення належить одному Користувачу і може містити багато Ефектів.

3) Кожне Зображення може входити до багатьох Колажів (багато-до-багатьох).

4) Кожен Колаж належить одному Користувачу та містить набір Зображень.

5) Кожен Ефект застосовується до одного Зображення.

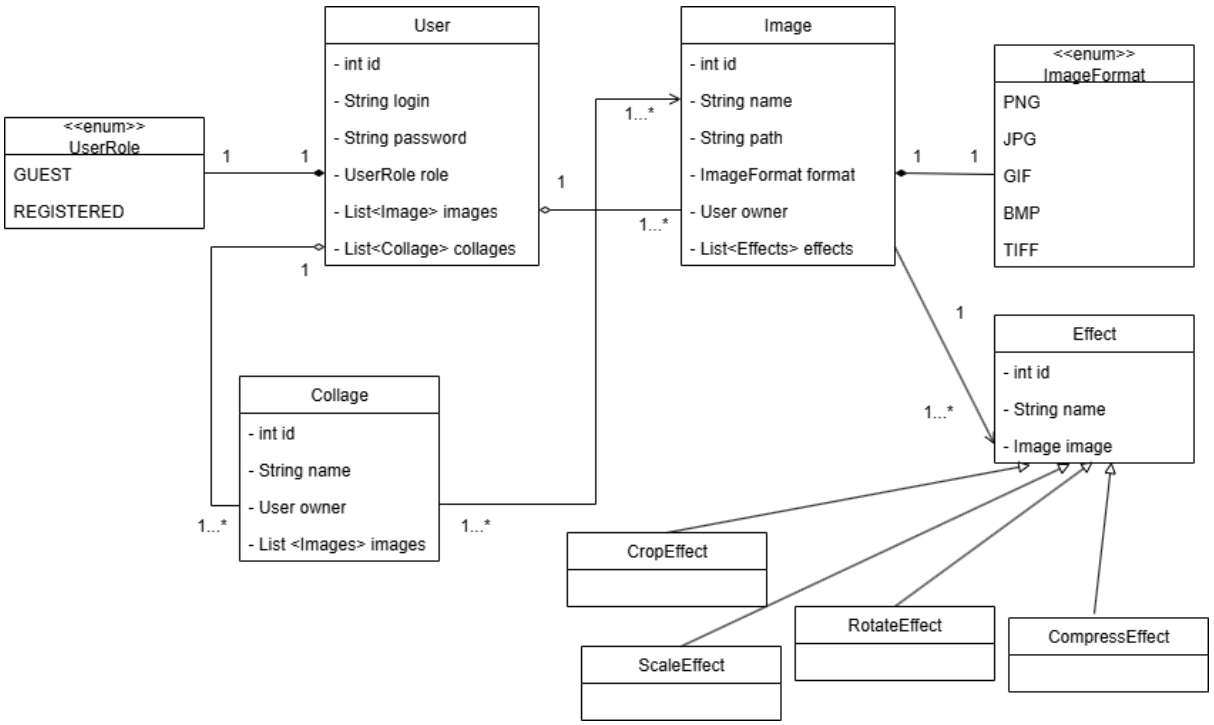


Рисунок 2. - Діаграма класів предметної області

Проектування структури бази даних та реалізація патерну Repository

Для зберігання метаданих про користувачів, зображення, ефекти та колажі було спроєктовано реляційну базу даних. Її структура відображає сутності, визначені в моделі предметної області, та забезпечує цілісність даних. Таблиця User зберігає інформацію про всіх користувачів системи, включаючи логін, пароль та роль. Таблиця Image містить деталі про кожне зображення, такі як назва, формат, шлях до файлу, розмір та роздільна здатність, а також посилання на власника. Таблиця Effect фіксує інформацію про застосовані до зображень трансформації, наприклад поворот, кадрування або стиснення, і пов’язана з конкретним зображенням. Таблиця Collage використовується для збереження композицій із декількох зображень і містить інформацію про власника та набір зображень.

Між таблицями реалізовані відношення типу «один-до-багатьох» та «багато-до-багатьох», що дозволяє відобразити бізнес-логіку системи: один користувач може мати багато зображень та колажів, одне зображення може мати багато ефектів і входити до кількох колажів, а кожен колаж містить набір зображень. Для взаємодії з цією базою даних реалізовано патерн проєктування Repository, який ізолює бізнес-логіку застосунку від конкретних деталей роботи з базою даних та забезпечує стандартні операції CRUD для всіх сутностей, а також можливість виконувати додаткові запити. Поєднання реляційної структури та Repository дозволяє ефективно керувати даними та реалізовувати всі функціональні можливості редактора зображень, включаючи відкриття і збереження файлів, застосування ефектів і створення колажів.

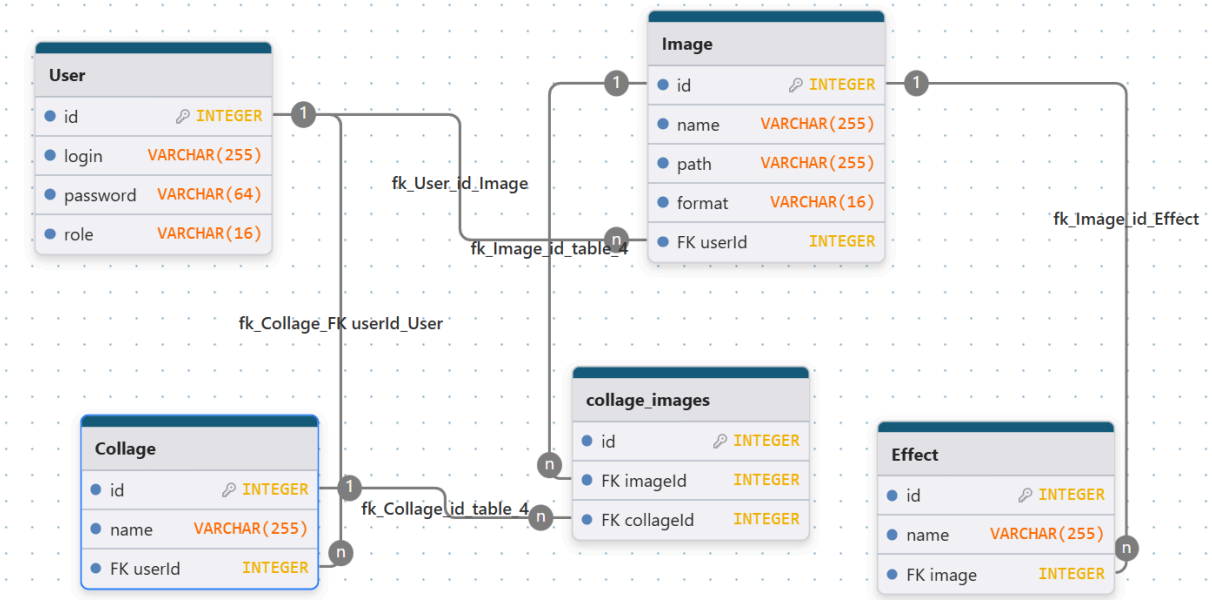


Рисунок 3. - Схема даних, реалізована в drawDB

Проектування діаграми класів реалізованої системи

Фінальна діаграма класів відображає повну архітектуру розробленої частини системи. Вона об'єднує класи предметної області, моделі даних, репозиторії та всі застосовані патерни проєктування, забезпечуючи надійну та розширювану структуру. Основні елементи фінальної діаграми: Сутності предметної області та їхні зв'язки. Репозиторії, що ізолюють логіку доступу до даних від бізнес-логіки.

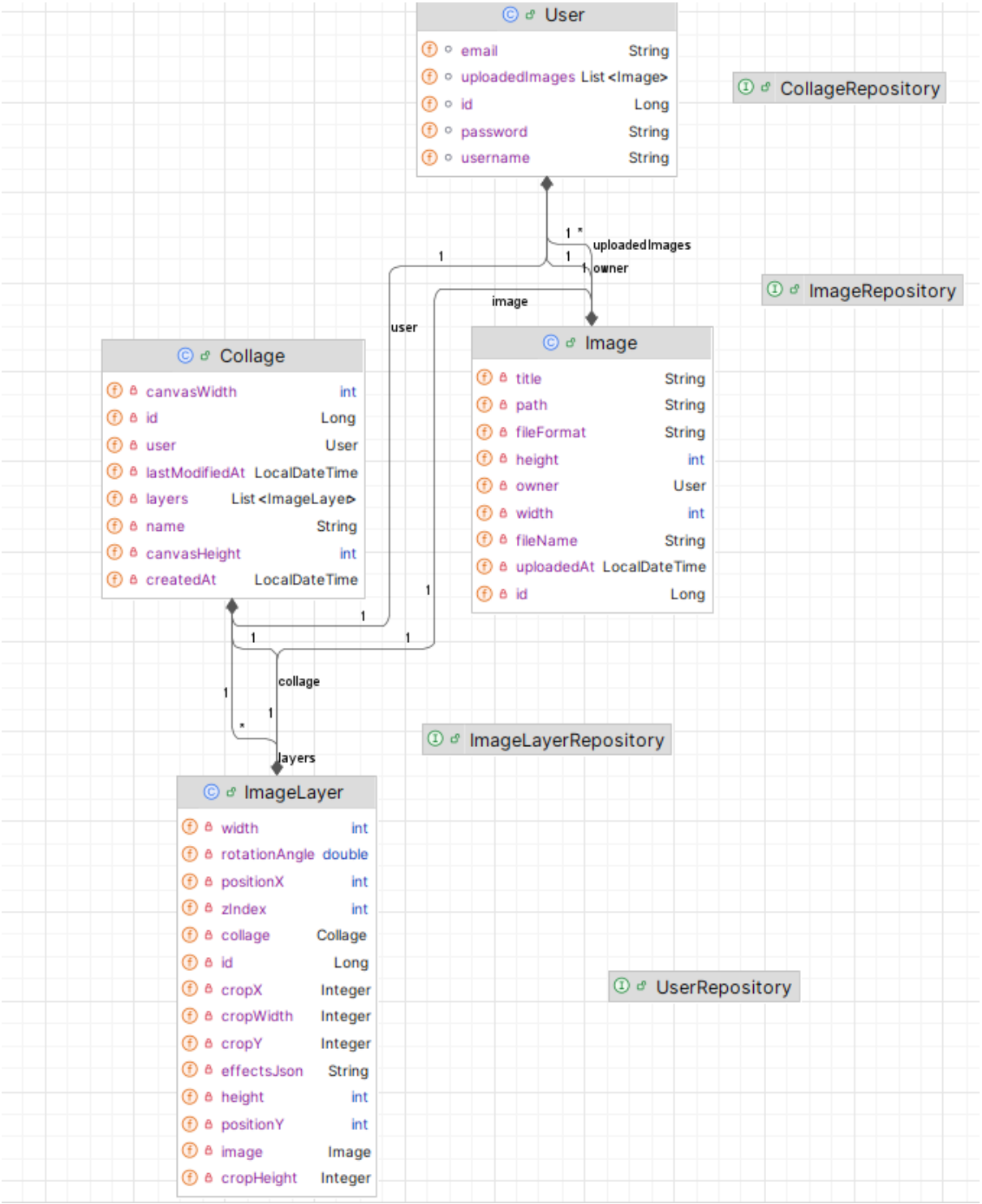
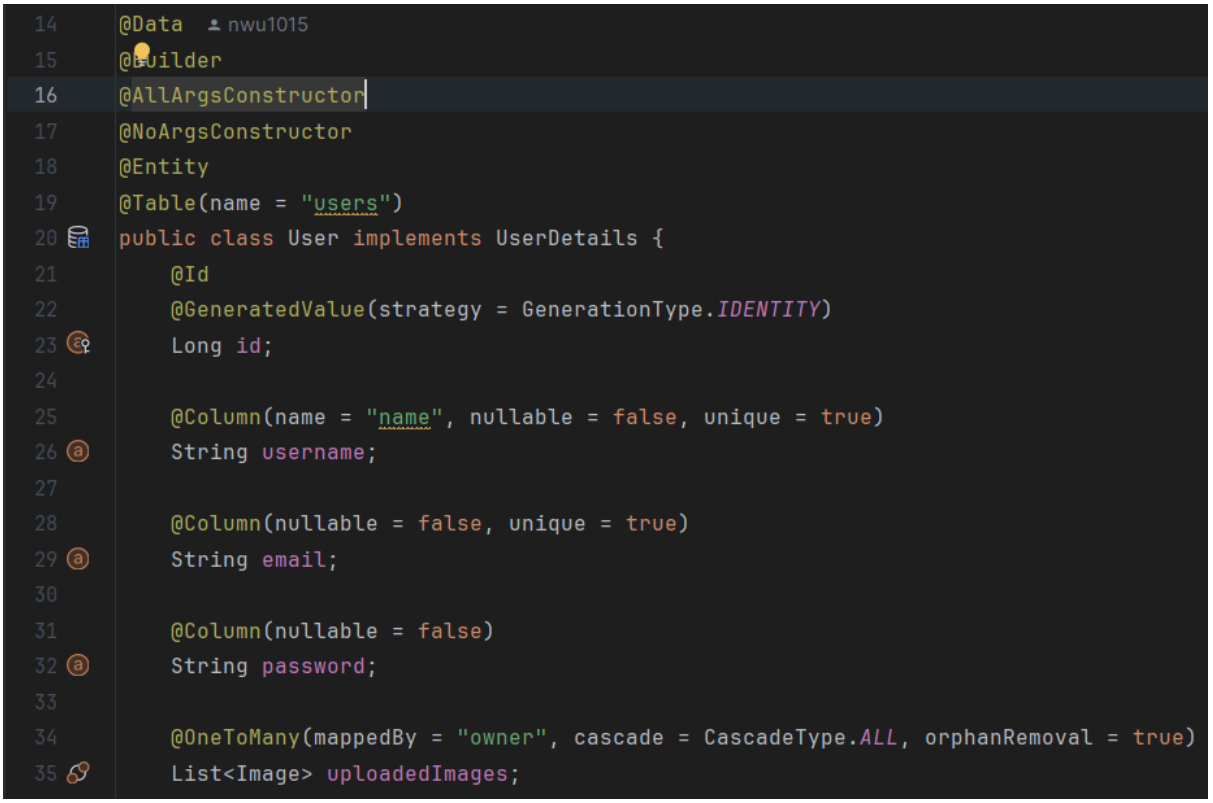


Рисунок 4. - Діаграми класів реалізованої системи



```java
@Data  nwu1015
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;

    @Column(name = "name", nullable = false, unique = true)
    String username;

    @Column(nullable = false, unique = true)
    String email;

    @Column(nullable = false)
    String password;

    @OneToMany(mappedBy = "owner", cascade = CascadeType.ALL, orphanRemoval = true)
    List<Image> uploadedImages;
```

Рисунок 5. - Програмний код класу User

```java
@Entity  nwu1015
@Table(name = "images")
@Data
@NoArgsConstructor
public class Image {

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

    @Column(nullable = false)
    private String fileName;

    @Column
    private String title;

    // Шлях до файлу у файловій системі або URL у хмарному сховищі
    @Column(nullable = false)
    private String path;

    @Column(nullable = false)
    private String fileFormat;

    private int width;
    private int height;

    @Column(updatable = false)
    private LocalDateTime uploadedAt = LocalDateTime.now();

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "owner_id")
    private User owner;
```

Рисунок 6. - Програмний код класу Image

```java
@Entity  nwu1015
@Table(name = "image_layers")
@Data
@NoArgsConstructor
public class ImageLayer {

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

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "project_id", nullable = false)
    private Collage collage;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "image_id", nullable = false)
    private Image image;

    //Властивості трансформації
    private int positionX;
    private int positionY;

    private int width;  // Ширина шару
    private int height; // Висота шару

    private double rotationAngle = 0.0; // Кут повороту в градусах

    private int zIndex; // Порядок шару (0 - найнижчий, вищі значення - ближче до глядача)
```

```java
    // Параметри кадрування
    private Integer cropX;
    private Integer cropY;
    private Integer cropWidth;
    private Integer cropHeight;

    @Column(columnDefinition = "TEXT")
    private String effectsJson;
}
```

Рисунок 7. - Програмний код класу ImageLayer

```java
@Entity  nwu1015
@Table(name = "collages")
@Data
@NoArgsConstructor
public class Collage {

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

    @Column(nullable = false)
    private String name;

    private int canvasWidth;
    private int canvasHeight;

    private LocalDateTime createdAt = LocalDateTime.now();
    private LocalDateTime lastModifiedAt = LocalDateTime.now();

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @OneToMany(mappedBy = "collage", cascade = CascadeType.ALL, orphanRemoval = true)
    @OrderBy("zIndex ASC")
    private List<ImageLayer> layers = new ArrayList<>();
}
```

Рисунок 8. - Програмний код класу Collage

```java
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);  2 usages  nwu1015
}
```

Рисунок 9. - Програмний код класу UserRepository

```java
public interface ImageRepository extends JpaRepository<Image, Long> {
    List<Image> findByOwner(User owner);  1 usage  nwu1015
}
```

Рисунок 10. - Програмний код класу ImageRepository

```java
public interface ImageLayerRepository extends JpaRepository<ImageLayer, Long> {
    Optional<ImageLayer> findById(Long id);  8 usages  nwu1015

    Optional<ImageLayer> findFirstByImageAndCollage_User(Image image, User user);

    List<ImageLayer> findAllByImage(Image image);  1 usage  nwu1015
}
```

Рисунок 11. - Програмний код класу ImageLayerRepository

```java
public interface CollageRepository extends JpaRepository<Collage, Long> {
}
```

Рисунок 12. - Програмний код класу CollageRepository

Висновок: У ході виконання роботи було спроєктовано та реалізовано модель предметної області для системи редагування зображень, що включає основні сутності – користувачів, зображення, ефекти та колажі – та визначає їхні взаємозв'язки. На основі цієї моделі була розроблена реляційна база даних, що забезпечує цілісність та ефективне зберігання метаданих, а також дозволяє масштабувати систему та додавати нові функціональні можливості. Для взаємодії з базою даних реалізовано паттерн Repository, який ізолює бізнес-логіку від деталей роботи з БД та забезпечує стандартизовані операції CRUD.

Завдяки такому підходу система підтримує основні функції редактора: відкриття і збереження зображень у різних форматах, застосування різноманітних ефектів, створення колажів і керування ними. Виконана робота демонструє логічне і структуроване подання даних, що полегшує подальший розвиток та підтримку системи, а також створює надійну основу для реалізації інтерфейсу користувача та додаткових сервісів. Таким чином, розроблена модель і база даних забезпечують ефективну, гнучку та зручну платформу для роботи з графічними об'єктами.

Питання до лабораторної роботи

1. Що таке UML?

Це уніфікована мова моделювання, яка використовується для візуального опису структури, поведінки та взаємодії компонентів програмних систем. UML дозволяє графічно показати, як працює система, без прив’язки до конкретної мови програмування чи технології бази даних. Це допомагає розробникам, аналітикам та замовникам зрозуміти логіку та архітектуру системи.

2. Що таке діаграма класів UML?

Це графічне представлення статичної структури системи, яке показує основні класи, їхні атрибути та методи, а також зв’язки між ними. Вона дозволяє зрозуміти, як різні частини системи взаємодіють одна з одною і як організовані дані. Діаграма класів відображає такі відносини, як наслідування, асоціації та композиції, а також множинність цих зв’язків. Використання такої діаграми допомагає розробникам і аналітикам планувати архітектуру системи, моделювати бізнес-логіку та проєктувати базу даних, не заглиблюючись у конкретні деталі реалізації.

3. Які діаграми UML називають канонічними?

Це стандартизовані типи діаграм, які дозволяють моделювати програмну систему як у структурі, так і в поведінці. Вони допомагають розробникам і аналітикам зрозуміти, як організовані дані, класи та взаємозв’язки між ними, а також як об’єкти взаємодіють у процесах і алгоритмах.

Структурні діаграми показують статичну архітектуру системи, включаючи класи, атрибути, методи та зв’язки між ними, а поведінкові діаграми відображають динамічну поведінку, процеси та взаємодію об’єктів у часі. Використання канонічних діаграм UML дозволяє

комплексно моделювати систему, спрощує проєктування, реалізацію та подальшу підтримку програмного забезпечення.

4. Що таке діаграма варіантів використання?

Це тип поведінкової діаграми, яка описує функціональність системи з точки зору користувача. Вона показує, які дії користувачі або інші системи можуть виконувати, а також взаємодію між користувачами та системою.

На такій діаграмі відображаються актор (той, хто використовує систему) і варіанти використання (Use Case), які демонструють основні функції системи. Зв’язки між акторами та варіантами використання показують, хто і яким чином взаємодіє із системою, а додаткові залежності можуть відображати розширення або включення функцій.

Діаграми варіантів використання допомагають зрозуміти що система повинна робити, не вдаючись до деталей реалізації, і слугують основою для планування функціональних вимог та тестування.

5. Що таке варіант використання?

Це опис конкретної функції або дії, яку система виконує для користувача або іншої системи. Він показує, як актор (користувач чи зовнішня система) взаємодіє із системою для досягнення певної мети.

Варіант використання визначає що система робить, а не як вона це реалізує, тобто фокусується на функціональних вимогах. Він може включати основний сценарій успішного виконання дії, а також альтернативні або виняткові сценарії.

6. Які відношення можуть бути відображені на діаграмі використання?

На діаграмі варіантів використання можуть бути відображені кілька типів відношень, які показують зв’язки між акторами і варіантами

використання, а також між самими варіантами використання. Основні з них включають:

Асоціація – показує прямий зв’язок між актором і варіантом використання, тобто хто використовує певну функцію системи.

Включення («include») – показує, що один варіант використання завжди виконує інший як частину своєї поведінки. Це допомагає уникати дублювання спільних функцій.

Розширення («extend») – показує необов’язкове або додаткове виконання іншого варіанта використання, яке відбувається лише за певних умов.

Загальні (Generalization) – застосовується до акторів або варіантів використання, коли один елемент наслідує поведінку іншого, тобто спеціалізує його.

7. Що таке сценарій?

Це опис послідовності дій або кроків, які виконуються під час використання системи для досягнення певної мети. Він деталізує, як актор взаємодіє із системою в рамках конкретного варіанту використання.

Сценарій може включати основний (базовий) потік подій, який описує успішне виконання дії, а також альтернативні або виняткові потоки, які показують, що відбувається у випадку помилок або нестандартних ситуацій.

Наприклад, для варіанту використання «Збереження зображення» основний сценарій може включати вибір файлу та натискання кнопки «Зберегти», а альтернативний сценарій може описувати, що відбувається, якщо файл не вдається зберегти через відсутність доступу до диска.

8. Що таке діаграма класів?

Це тип структурної діаграми UML, який показує основні класи системи, їхні атрибути та методи, а також зв’язки між ними. Вона дозволяє візуально

відобразити статичну структуру програмного забезпечення, тобто як організовані дані та об’єкти, і які відносини існують між ними.

Діаграма класів допомагає розробникам і аналітикам зрозуміти, які компоненти входять до системи, як вони взаємодіють і яку інформацію зберігають. Вона відображає такі відношення, як асоціації, композиції, агрегації та наслідування, а також множинність зв’язків між класами.

9. Які зв’язки між класами ви знаєте?

Асоціація відображає загальний зв’язок між класами, коли один клас використовує або посилається на інший. Вона може мати множинність, що показує, скільки об’єктів одного класу може бути пов’язано з об’єктами іншого.

Агрегація описує відношення «ціле–частина», де один клас складається з інших, але частини можуть існувати самостійно. Композиція — це більш жорстка форма агрегації, де частини не можуть існувати без цілого.

Наслідування (Generalization) показує, що один клас успадковує властивості та методи іншого, тобто спеціалізує його поведінку.

Залежність (Dependency) відображає тимчасовий або слабкий зв’язок, коли один клас використовує інший лише для виконання певної операції або методу, без постійного збереження посилання на нього.

10. Чим відрізняється композиція від агрегації?

У агрегації частина може існувати незалежно від цілого. Наприклад, клас «Класна кімната» може мати об’єкти «Столи», але столи можуть існувати самостійно поза цією кімнатою. Агрегація показує слабкий зв’язок між класами.

У композиції частина не може існувати без цілого. Наприклад, клас «Автомобіль» складається з «Двигуна», і двигун не має сенсу існувати

окремо без автомобіля. Композиція показує сильний, власницький зв’язок між класами, і знищення цілого призводить до знищення всіх його частин.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів?

Агрегація показує слабкий зв’язок «ціле–частина», де частина може існувати незалежно від цілого. На діаграмі вона позначається порожнім ромбом на боці цілого класу. Це означає, що ці об’єкти пов’язані, але їхнє існування не взаємозалежне.

Композиція показує сильний, власницький зв’язок, де частина не може існувати без цілого. На діаграмі вона позначається заповненим ромбом на боці цілого класу. Це означає, що знищення цілого призводить до знищення всіх його частин, і частини повністю залежні від цілого.

12. Що являють собою нормальні форми баз даних?

Нормальні форми баз даних — це правила або критерії організації таблиць у реляційній базі даних, які покликані зменшити надлишковість даних і уникнути аномалій при їхньому оновленні, видаленні або вставці. Вони допомагають забезпечити логічну цілісність даних та спрощують їхню структуру.

Кожна нормальна форма вводить певні обмеження. Наприклад, перша нормальна форма (1NF) вимагає, щоб усі поля таблиці містили атомарні, тобто неподільні значення. Друга нормальна форма (2NF) вимагає, щоб усі неключові атрибути повністю залежали від первинного ключа. Третя нормальна форма (3NF) вимагає відсутності транзитивних залежностей між атрибутами. Існують також більш високі нормальні форми, які додатково оптимізують структуру даних для складних випадків.

Використання нормальних форм дозволяє створювати ефективну та гнучку базу даних, де дані зберігаються структуровано, зменшується дублювання і підвищується надійність роботи системи.

13. Що таке фізична модель бази даних? Логічна?

Логічна модель бази даних описує дані та їхні взаємозв’язки незалежно від конкретної системи управління базами даних (СУБД). Вона включає сутності, атрибути, ключі та зв’язки між таблицями, а також правила цілісності даних. Логічна модель допомагає зрозуміти які дані зберігаються і як вони взаємодіють, не заглиблюючись у технічні деталі реалізації.

Фізична модель бази даних показує, як логічна структура реалізується конкретно в СУБД. Вона враховує типи даних, індекси, обмеження, розміщення таблиць на диску та оптимізацію продуктивності. Фізична модель визначає конкретне зберігання і організацію даних, необхідну для роботи системи в реальному середовищі.

14. Який взаємозв'язок між таблицями БД та програмними класами?

Взаємозв’язок між таблицями бази даних і програмними класами полягає в тому, що клас у програмі часто відображає одну таблицю в базі даних. Кожен об’єкт класу відповідає рядку таблиці, а атрибути класу – стовпцям таблиці.

Таке відображення дозволяє програмі працювати з даними бази як з об’єктами, приховуючи низькорівневі деталі SQL-запитів та зберігання. Зв’язки між класами, наприклад асоціації чи композиції, відображаються у базі даних як зовнішні ключі, проміжні таблиці або обмеження цілісності, що підтримують логічні взаємозв’язки між рядками таблиць.

У сучасних фреймворках, таких як Hibernate або Spring Data JPA, ці відповідності реалізуються через ORM (Object-Relational Mapping), що

автоматично синхронізує об’єкти класів і записи таблиць, дозволяючи розробникам працювати з даними в об’єктно-орієнтованому стилі.