Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 2

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Основи проектування»

Виконала
Студентка групи ІА-31: Перевірив:
Трегуб К. В. Мягкий М. Ю.

Київ 2025

1. **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.

2. 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. Мова UML є досить строгим та потужним засобом моделювання, який може бути ефективно використаний для побудови концептуальних, логічних та графічних 18 моделей складних систем різного цільового призначення. Ця мова увібрала в себе найкращі якості та досвід методів програмної інженерії, які з успіхом використовувалися протягом останніх років при моделюванні великих та складних систем.

Діаграма – це 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. Діаграма варіантів використання – це вихідна концептуальна модель проєктованої системи, вона не описує внутрішню побудову системи.

Діаграма використання складається з:

- Акторів - будь-які об'єкти, суб'єкти чи системи, що взаємодіють з модельованою бізнес-системою ззовні для досягнення своїх цілей або вирішення певних завдань.

- Варіантів використання - служать для опису служб, які система надає актору.

- Відношень: асоціація, узагальнення, залежність, включення, розширення. 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. Діаграма класів містить у собі класи, їхні методи та атрибути, зв’язки. Методи та атрибути мають 4 модифікатори доступу: public, package, protected, private. Зв’язки налічують у собі асоціацію, агрегацію, композицію, успадкування тощо

3. **Завдання:**
   - Ознайомитись з короткими теоретичними відомостями.
   - Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
   - Спроєктувати діаграму класів предметної області.
   - Вибрати 3 варіанти використання та написати за ними сценарії використання.
   - На основі спроєктованої діаграми класів предметної області розробити основні класи та структуру бази даних системи. Класи даних повинні реалізувати шаблон Repository для взаємодії з базою даних.
   - Нарисувати діаграму класів для реалізованої частини системи.
   - Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму варіантів використання відповідно, діаграму класів системи, вихідні коди класів системи, а також зображення структури бази даних

4. Хід роботи:

Тема: Сервіс бронювання послуг для салону краси.

Діаграма варіантів використання

Актори, варіанти використання та відношення:

Гість

- Переглянути список послуг
- Зареєструватися
- Авторизуватися

Клієнт

- Переглянути список послуг
- Створити бронювання
  <<include>> Переглянути розклад майстрів
  <<include>> Вибрати послугу
  <<extend>> Оплата бронювання онлайн
  <<extend>> Скасування оплати
- Залишити відгук про майстра
- Переглянути/скасувати бронювання

Майстер

- Позначення виконаних бронювань

Адміністратор

- Призначити розклад роботи майстру
- Додати/видалити/змінити майстра
- Додати/редагувати/видалити послугу

Система оплати

- Оплата бронювання онлайн
- Скасування оплати

Також між акторами є відношення узагальнення, вони успадковують атрибути один одного.

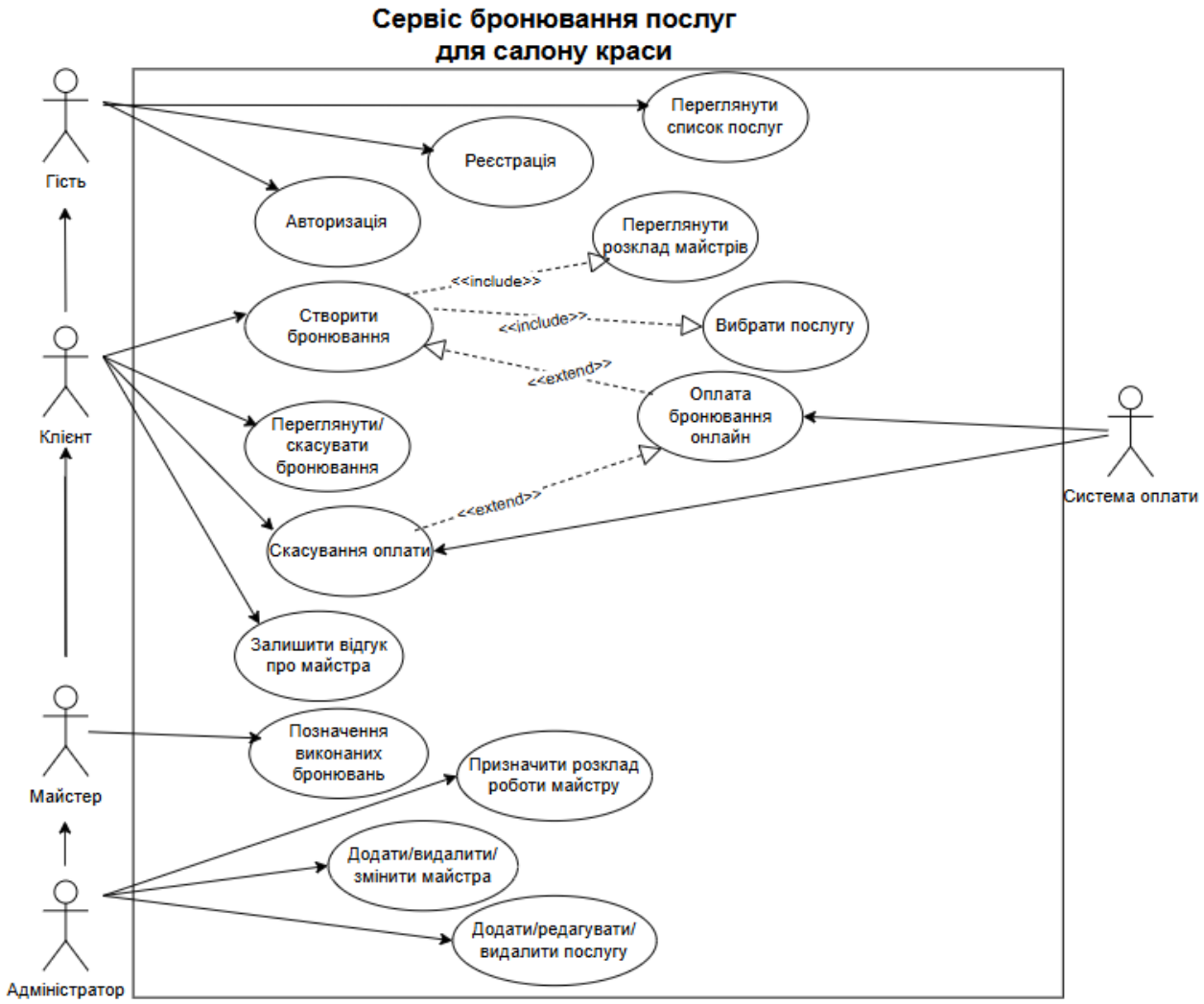

Рис. 1 – Діаграма варіантів використання

Сценарії використання

1. Створення бронювання

Передумови:
Клієнт авторизований у системі; салон має доступні послуги та вільних майстрів у розкладі.

Постумови:
Створено нове бронювання, яке збережене у системі та прив’язане до конкретного майстра, дати й часу.

Взаємодіючі сторони:
Клієнт, Сервіс бронювання послуг салону краси.

Короткий опис:
Цей сценарій визначає процес створення нового бронювання клієнтом.

Основний потік подій:

1. Клієнт переглядає список послуг.
2. Клієнт вибирає бажану послугу.
3. Система відображає доступних майстрів і вільні часові слоти.

4. Клієнт обирає майстра та дату з часом.
5. Система формує попереднє бронювання.
6. Клієнт підтверджує створення бронювання.
7. Система зберігає бронювання і повідомляє про успіх.

Винятки:

- Обраний майстер недоступний у вибраний час — система повідомляє про неможливість бронювання, клієнт повертається до вибору часу.
- Відсутні активні послуги — система повідомляє про помилку.

Примітки:
Клієнт може одразу перейти до оплати (варіант використання «Оплата бронювання онлайн»).

2. Оплата бронювання онлайн

Передумови:
Клієнт має створене бронювання; підключено систему електронних платежів.

Постумови:
Оплата виконана успішно; статус бронювання змінено на «оплачене».

Взаємодіючі сторони:
Клієнт, Сервіс бронювання послуг салону краси, Система оплати.

Короткий опис:
Цей сценарій визначає процес онлайн-оплати створеного бронювання.

Основний потік подій:

1. Клієнт відкриває сторінку своїх бронювань.
2. Клієнт обирає бронювання для оплати.
3. Система пропонує способи оплати.
4. Клієнт вводить платіжні дані та підтверджує платіж.
5. Система оплати перевіряє й обробляє платіж.
6. Система бронювання отримує підтвердження від системи оплати.
7. Статус бронювання змінюється на «оплачено».
8. Система повідомляє клієнта про успішну оплату.

Винятки:

- Недостатньо коштів або помилка платіжної системи — система повідомляє клієнта про невдалу оплату, статус бронювання залишається «неоплачено».
- Відсутнє активне бронювання — система повідомляє про неможливість оплати.

Примітки:
Варіант використання *«Оплата бронювання онлайн»* розширює (*extend*) варіант *«Створення бронювання»*.

3. Керування розкладом майстра

Передумови:
Адміністратор авторизований у системі; у базі є зареєстровані майстри та перелік послуг.

Постумови:
Розклад майстра створено або оновлено; зміни збережено у системі.

Взаємодіючі сторони:
Адміністратор, Система бронювання послуг салону краси.

Короткий опис:
Цей сценарій визначає процес створення або редагування розкладу роботи майстра адміністратором салону.

Основний потік подій:

1. Адміністратор відкриває розділ «Майстри».
2. Система відображає список зареєстрованих майстрів.
3. Адміністратор обирає потрібного майстра.
4. Система відображає поточний розклад роботи майстра.
5. Адміністратор додає або змінює робочі дні та часові проміжки.
6. Адміністратор підтверджує зміни.
7. Система оновлює розклад і повідомляє про успішне збереження.

Винятки:

- Майстер не знайдений — система повідомляє про помилку.
- Часові проміжки перетинаються або некоректні — система повідомляє про неможливість збереження і пропонує виправити дані.

Примітки:
Варіант використання *«Керування розкладом майстра»* включає (*include*) варіант *«Перегляд розкладу майстрів»*.

Структура БД:

- **users** - зберігає основну інформацію про всіх користувачів системи (клієнтів, майстрів та адміністраторів). Містить контактні дані, логін-дані та роль користувача.
- **services** - містить дані про послуги, які надає салон: назву, опис, ціну та тривалість процедури.
- **masters** - пов’язана з таблицею користувачів. Зберігає професійні дані майстрів: спеціалізацію та досвід роботи.
- **schedules** - описує робочі графіки майстрів, включаючи дати та робочі години.
- **bookings** - головна таблиця, що відображає процес бронювання послуг клієнтами. Містить інформацію про клієнта, майстра, обрану послугу, час запису, статус бронювання та вартість.
- **payments** - фіксує дані про оплату кожного бронювання: суму, спосіб, статус та дату платежу.

Зв’язки між таблицями

- Кожен **майстер** прив’язаний до одного **користувача** (masters.user_id > users.id).
- Кожен **розклад** належить певному **майстру** (schedules.master_id > masters.id).

- Кожне **бронювання** пов’язує **клієнта**, **майстра**, **послугу** та **розклад**.
- Кожна **оплата** стосується одного **бронювання** (payments.booking_id > bookings.id).

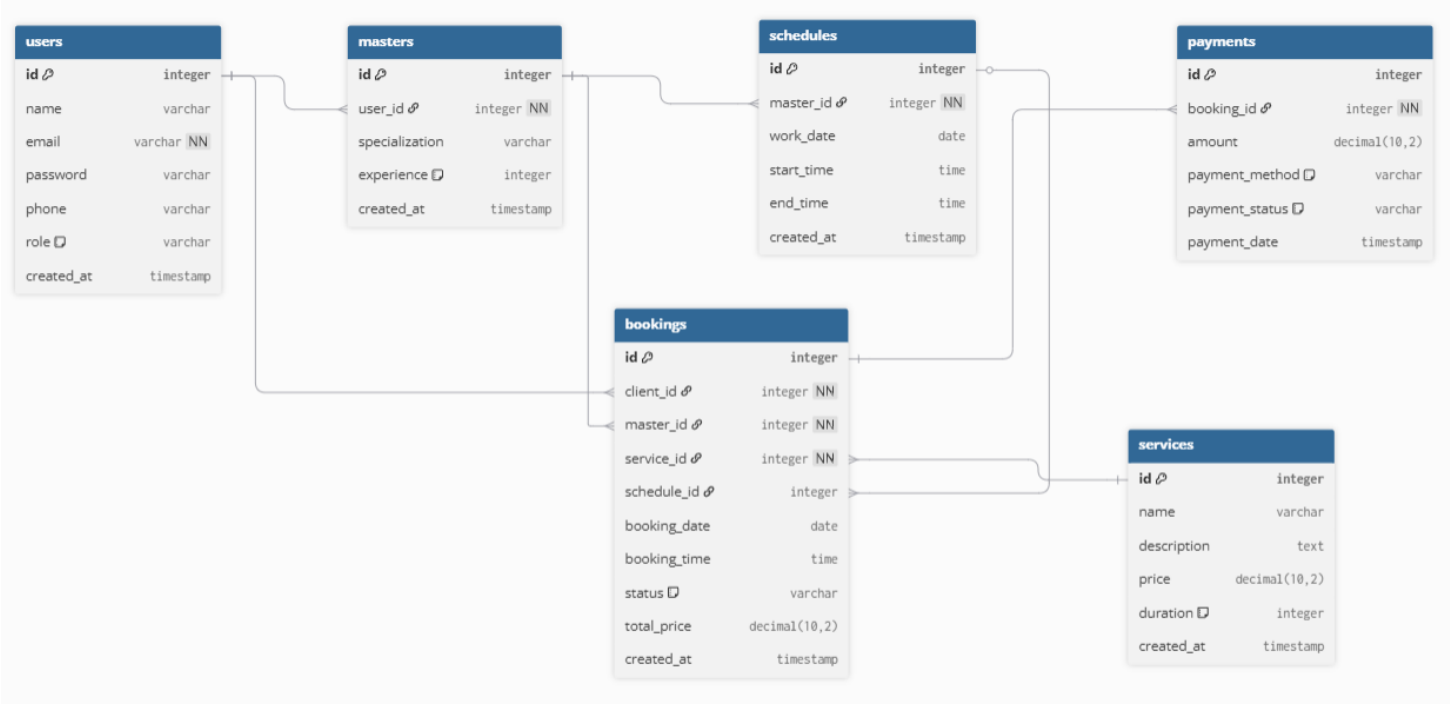

Рис. 2 – Структура бази даних

Діаграми класів:

1. Загальний інтерфейс IRepository<T, TKey>

- Узагальнений інтерфейс, що визначає базові CRUD-операції: отримання, додавання, оновлення та видалення сутностей.
- Методи, як-от GetById, AddEntity, SaveChangesAsync, уніфікують роботу з будь-якими об’єктами бази даних.

2. Абстрактний клас RepositoryBase<T, TKey>

- Реалізує загальну логіку для конкретних репозиторіїв.
- Служить базою, від якої наслідуються всі конкретні репозиторії.

3. Сутності (Entities)

- Представляють ключові об’єкти системи бронювання (користувачів, майстрів, послуги, розклад, замовлення, оплату).

4. Конкретні репозиторії

- Для кожної сутності існує свій репозиторій: UserRepository, MasterRepository, ServiceRepository тощо.
- Вони наслідують базову логіку з RepositoryBase та реалізують відповідні інтерфейси (IUserRepository, IMasterRepository і т.д.).

5. Інтерфейси конкретних репозиторіїв

- Кожен спеціалізований інтерфейс (IUserRepository, IBookingRepository тощо) розширює базовий IRepository.

6. Взаємозв'язки

- Всі конкретні репозиторії наслідують RepositoryBase.
- Кожен інтерфейс конкретного репозиторію наслідує IRepository.

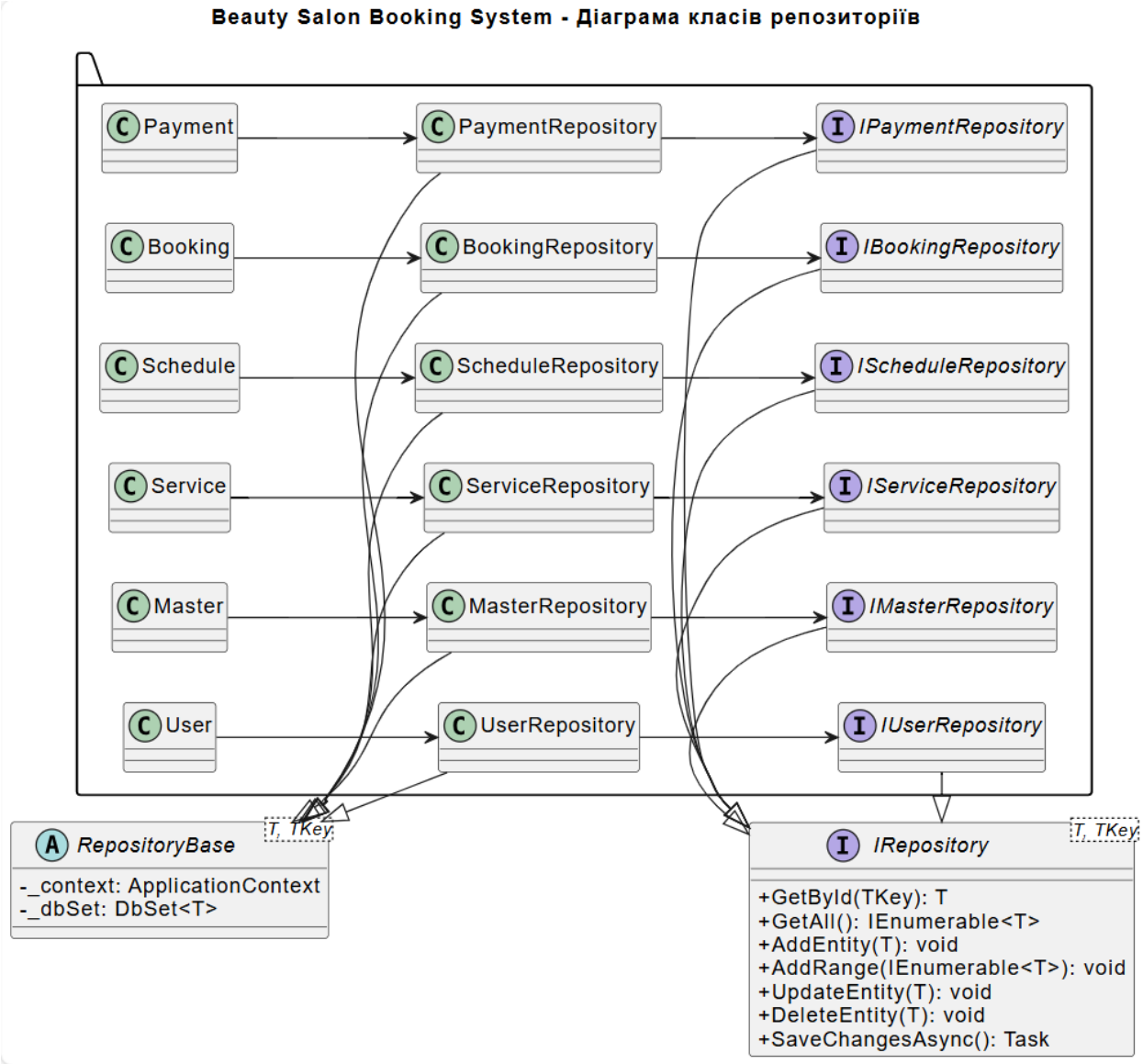

Рис. 3 – Діаграма класів репозиторіїв

User — представляє клієнта системи. Користувач може створювати бронювання, здійснювати оплату та залишати відгуки про отримані послуги.

Master — майстер, який пропонує послуги, має власний розклад роботи, виконує бронювання клієнтів.

Service — описує конкретну послугу салону (назву, тривалість і ціну). Кожна послуга пов'язана з майстром, який її надає, і може бути забронювана у межах бронювання.

Schedule — визначає розклад майстра, тобто дати та часові проміжки, коли він доступний для записів. Один розклад може включати кілька бронювань.

Booking — центральний клас системи, який об’єднує користувача, майстра, послугу та розклад. Кожне бронювання має оплату.

Payment — представляє оплату за конкретне бронювання. Зв’язок між Booking і Payment є композицією, оскільки оплата не існує без бронювання.

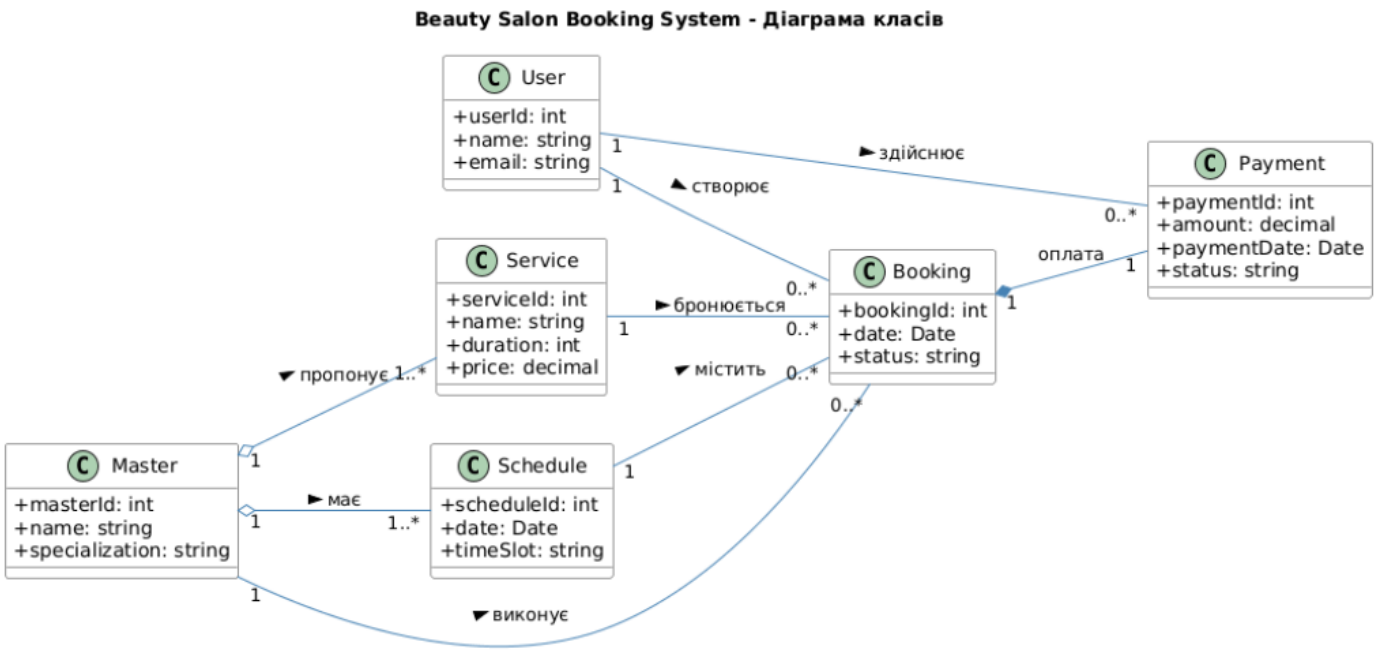

Рис. 4 – Діаграма класів

Вихідний код класів:

Універсальний інтерфейс репозиторію

```
package com.beautysalon.data;

import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;

public interface IRepository<T, TKey> {
    Optional<T> getById(TKey id);
    List<T> getAll();
    void addEntity(T entity);
    void updateEntity(T entity);
    void deleteEntity(TKey id);
    CompletableFuture<Void> saveChangesAsync();
}
```

Рис. 5 – Код інтерфейсу IRepository<T, TKey>

Базовий клас репозиторію

```java
package com.beautysalon.data;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import jakarta.transaction.Transactional;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;

@Transactional
public abstract class RepositoryBase<T, TKey> implements IRepository<T, TKey> {

    @PersistenceContext
    protected EntityManager entityManager;
    private final Class<T> entityClass;

    protected RepositoryBase(Class<T> entityClass) {
        this.entityClass = entityClass;
    }

    @Override
    public Optional<T> getById(TKey id) {
        return Optional.ofNullable(entityManager.find(entityClass, id));
    }

    @Override
    public List<T> getAll() {
        return entityManager.createQuery("from " + entityClass.getSimpleName(), entityClass).getResul
    }

    @Override
    public void addEntity(T entity) {
        entityManager.persist(entity);
    }

    @Override
    public void updateEntity(T entity) {
        entityManager.merge(entity);
    }

    @Override
    public void deleteEntity(TKey id) {
        getById(id).ifPresent(entityManager::remove);
    }

    @Override
    public CompletableFuture<Void> saveChangesAsync() {
        return CompletableFuture.runAsync(() -> entityManager.flush());
```

Рис. 5,6 – Код класу RepositoryBase<T, TKey>

Сутності (Entities)

```java
@Entity
public class User {
    @Id
    @GeneratedValue
    private UUID userId;
    private String name;
    private String email;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Booking> bookings;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Payment> payments;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Review> reviews;

}
```

Рис. 7 – Код класу User

```java
@Entity
public class Master {
    @Id
    @GeneratedValue
    private UUID masterId;
    private String name;
    private String specialization;

    @OneToMany(mappedBy = "master", cascade = CascadeType.ALL)
    private List<Service> services;

    @OneToMany(mappedBy = "master", cascade = CascadeType.ALL)
    private List<Schedule> schedules;

    @OneToMany(mappedBy = "master", cascade = CascadeType.ALL)
    private List<Booking> bookings;

    @OneToMany(mappedBy = "master", cascade = CascadeType.ALL)
    private List<Review> reviews;
```

Рис. 8 – Код класу Master

```java
@Entity
public class Service {
    @Id
    @GeneratedValue
    private UUID serviceId;
    private String name;
    private int durationMinutes;
    private double price;

    @ManyToOne
    @JoinColumn(name = "master_id")
    private Master master;

    @OneToMany(mappedBy = "service", cascade = CascadeType.ALL)
    private List<Booking> bookings;
}
```

Рис. 9 – Код класу Service

```java
@Entity
public class Schedule {
    @Id
    @GeneratedValue
    private UUID scheduleId;
    private LocalDateTime date;
    private String timeSlot;

    @ManyToOne
    @JoinColumn(name = "master_id")
    private Master master;

    @OneToMany(mappedBy = "schedule", cascade = CascadeType.ALL)
    private List<Booking> bookings;
}
```

Рис. 10 – Код класу Schedule

```java
public class Booking {
    @Id
    @GeneratedValue
    private UUID bookingId;
    private LocalDateTime date;
    private String status;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    @ManyToOne
    @JoinColumn(name = "master_id")
    private Master master;

    @ManyToOne
    @JoinColumn(name = "service_id")
    private Service service;

    @ManyToOne
    @JoinColumn(name = "schedule_id")
    private Schedule schedule;

    @OneToOne(mappedBy = "booking", cascade = CascadeType.ALL)
    private Payment payment;

    @OneToOne(mappedBy = "booking", cascade = CascadeType.ALL)
    private Review review;
}
```

Рис. 11 – Код класу Booking

```java
@Entity
public class Payment {
    @Id
    @GeneratedValue
    private UUID paymentId;
    private double amount;
    private LocalDateTime paymentDate;
    private String status;

    @OneToOne
    @JoinColumn(name = "booking_id")
    private Booking booking;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
```

Рис. 12 – Код класу Payment

5. Висновок

У процесі виконання даної лабораторної роботи на тему «Сервіс бронювання послуг для салону краси» було проведено комплексний аналіз вимог і спроєктовано структуру майбутньої системи. За допомогою UML було створено діаграми варіантів використання, діаграму класів, діаграму репозиторіїв та модель структури бази даних, які забезпечують цілісне бачення архітектури майбутнього застосунку.

Під час розроблення застосовано сучасні підходи програмної інженерії — зокрема, патерн Repository, який підвищує гнучкість і модульність коду, а також асинхронне програмування, що забезпечує ефективну обробку запитів і підвищує продуктивність системи.

Отримані результати дозволили сформувати не лише теоретичну модель майбутнього сервісу, а й основу для його подальшої реалізації. Створена структура забезпечує легке розширення функціоналу, спрощує підтримку та може бути використана як фундамент для розроблення повноцінного веб-додатку бронювання послуг у салоні краси.

Питання до лабораторної роботи

1. Що таке UML? – універсальна мова для опису та візуалізації систем.

2. Що таке діаграма класів UML? – схема з класами, їхніми полями, методами та зв’язками.

3. Які діаграми UML називають канонічними? – стандартні: класів, об’єктів, варіантів використання, послідовностей тощо.

4. Що таке діаграма варіантів використання? – показує акторів і функції, які вони виконують у системі.

5. Що таке варіант використання? – дія або функція, яку виконує користувач.

6. Які відношення можуть бути відображені на діаграмі використання? – асоціація, include, extend, узагальнення.

7. Що таке сценарій? – опис кроків взаємодії користувача з системою.

8. Що таке діаграма класів? – схема структури системи через класи та їх зв’язки.

9. Які зв’язки між класами ви знаєте? – асоціація, агрегація, композиція, наслідування, залежність.

10. Чим відрізняється композиція від агрегації? – у композиції частини не існують без цілого, в агрегації можуть.

11. Чим відрізняється зв'язки типу агрегації від зв'язків композиції на діаграмах класів? – агрегація має порожній ромб, композиція – зафарбований.

12. Що являють собою нормальні форми баз даних? – правила організації таблиць для уникнення дублювання.

13. Що таке фізична модель бази даних? Логічна? – фізична: як реально зберігається; логічна: концептуальна схема.

14. Який взаємозв'язок між таблицями БД та програмними класами? – таблиця ≈ клас, рядок ≈ об’єкт, стовпчик ≈ поле.