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»

Варіант - 17

Виконав
Студент групи ІА-31:
Лисенко В. Є

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

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:**

Мій варіант:

17. **System activity monitor** (iterator, command, abstract factory, bridge, visitor,

SOA) Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Діаграма варіантів використання

Рис. 1 - Діаграма використання

Актори:

1. Користувач (User): Людина, за якою ведеться спостереження (або яка переглядає свою статистику).
2. Адміністратор (Admin): Налаштовує параметри моніторингу (частоту опитування CPU, заборонені програми).
3. Системний Таймер (System Timer): (Це важливо для твоєї теми) Внутрішній актор, який автоматично ініціює збір даних щосекунди.

Варіанти використання (Use Cases):

- Основні:
    - Запуск моніторингу.
    - Зупинка моніторингу.
    - Формування звіту (Daily/Weekly).
- Автоматичні (Include):
    - Запис показників ресурсів (CPU/RAM).
    - Запис активності вводу (Keyboard/Mouse).

- Фіксація активного вікна.
- Визначення простою (Idle detection).

- Керування:
  - Налаштування інтервалів оновлення.
  - Експорт статистики (PDF/Excel).

Сценарії використання:

Табл. 1 Сценарій запис показників ресурсів

Елемент	Опис
Передумови	Моніторинг запущено, таймер спрацював (тік).
Постумови	У базі даних створено новий запис про навантаження.
Взаємодіючі сторони	Системний Таймер, Модуль збору даних.
Короткий опис	Система зчитує поточний стан обладнання та зберігає його.
Основний потік	1. Таймер ініціює подію оновлення. 2. Система запитує відсоток завантаження CPU. 3. Система запитує обсяг вільної RAM. 4. Дані формуються в об'єкт ResourceSnapshot. 5. Об'єкт передається в репозиторій для збереження.
Винятки	Помилка доступу до лічильників продуктивності (запис в лог помилок).

Табл. 2 Сценарій визначення простою (Idle State)

Елемент	Опис
Передумови	Моніторинг активний.
Постумови	Статус сесії змінюється на "Active" або "Idle".
Взаємодіючі сторони	Користувач, Input Monitor.
Короткий опис	Система перевіряє, як довго не було дій мишею чи клавіатурою.
Основний потік	1. Система слухає глобальні хуки (hooks) клавіатури/миші. 2. Якщо подія сталася -> скинути таймер простою. 3. Якщо таймер перевищив поріг (наприклад, 5 хв) -> створити подію IdleStarted. 4. При наступній дії -> створити подію IdleEnded.

Табл. 3 Сценарій формування звіту про активність

Елемент	Опис
Передумови	У базі є накопичені дані за обраний період.
Постумови	Користувач отримує візуалізовану статистику.
Взаємодіючі сторони	Користувач, Report Generator.
Основний потік	1. Користувач обирає період (наприклад, "Сьогодні"). 2. Система витягує всі записи ActivitySession. 3. Система групує дані по годинах. 4. Розраховується середній % CPU та час активності. 5. Результат виводиться на екран.

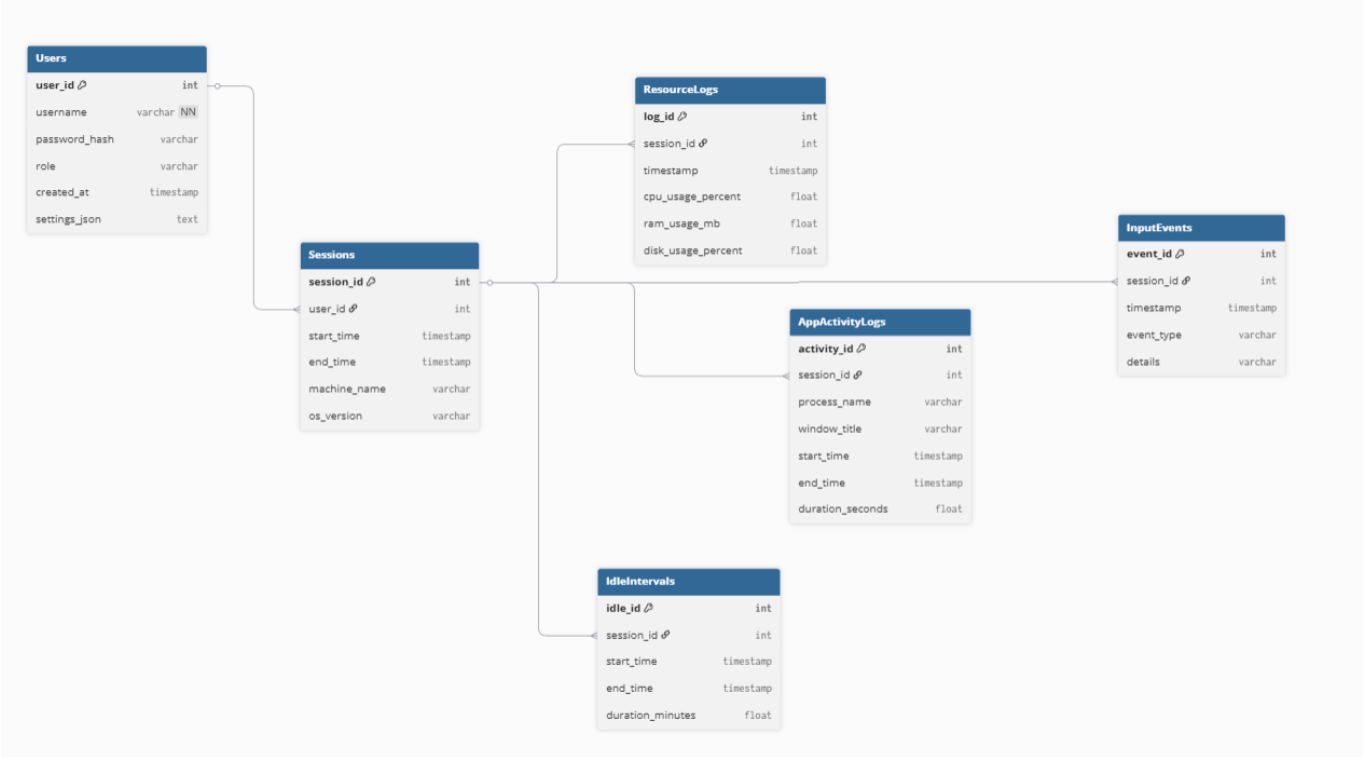

Рис. 2 - Структура БД

1. Users: (UserId, Name, Preferences) — хто користується.
2. Sessions: (SessionId, UserId, StartTime, EndTime) — період роботи програми (від запуску до закриття).
3. ResourceLogs: (LogId, SessionId, Timestamp, CpuUsagePercent, RamUsageMb) — "сирі" дані кожну секунду/хвилину.
4. ActivityEvents: (EventId, SessionId, EventType [Mouse/Key/Window], Details, Timestamp) — конкретні дії.
5. AppUsages: (AppId, SessionId, AppName, DurationSeconds) — скільки часу яка програма була активною.

Зв'язки: One-to-Many від Users до Sessions. One-to-Many від Sessions до всіх інших таблиць.

Діаграми класів:

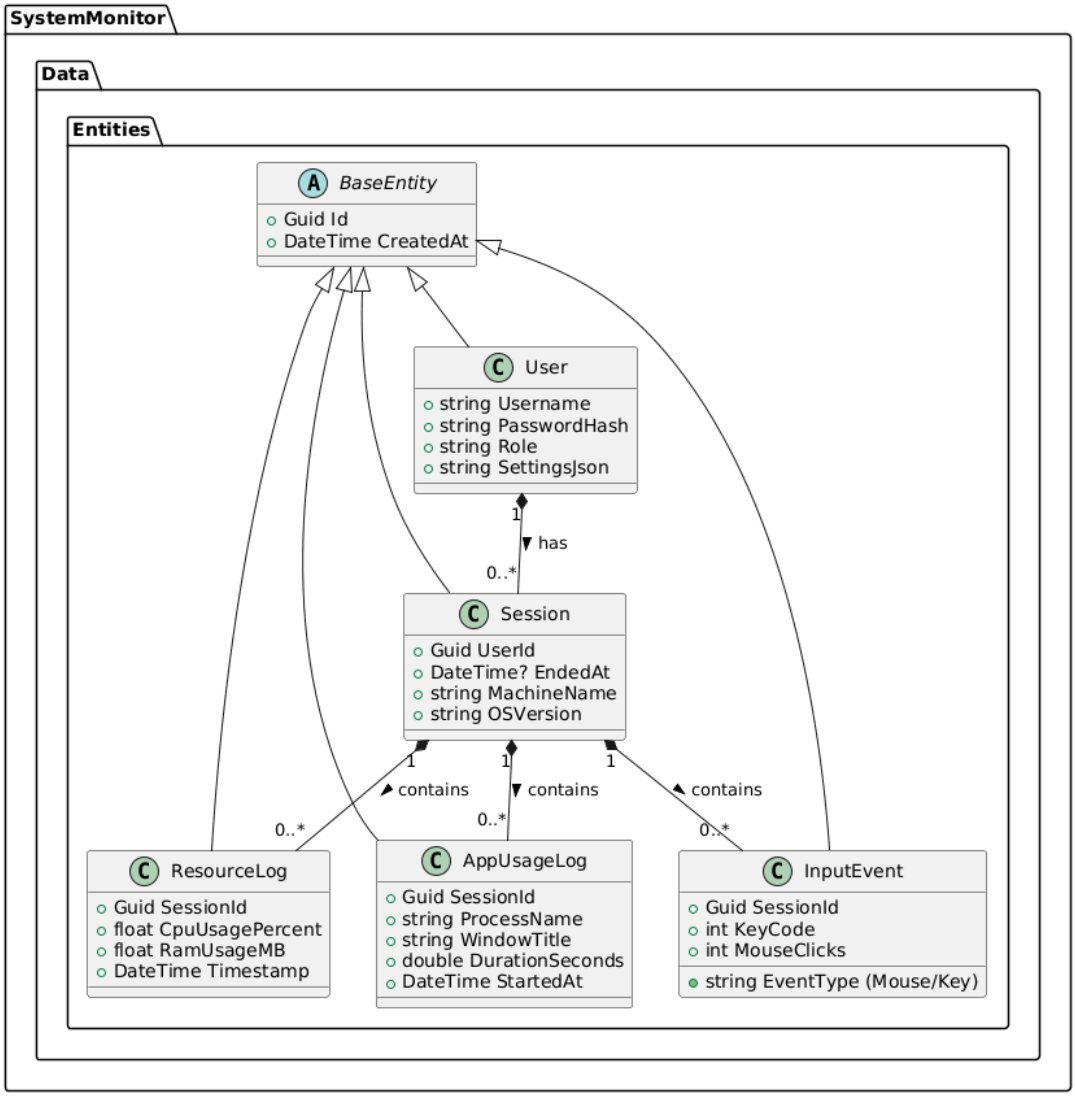

Рис. 3 - Діаграма класів

Діаграма, зображена на рисунку 3 відображає статичну структуру предметної області. Розроблено об'єктно-орієнтовану модель, що базується на принципі наслідування.

- BaseEntity: Абстрактний базовий клас, що містить спільні атрибути для всіх сутностей (Id, CreatedAt). Це дозволяє уникнути дублювання коду та забезпечує уніфікацію первинних ключів.
- User: Представляє обліковий запис користувача. Містить налаштування моніторингу у форматі JSON.
- Session: Ключова сутність, що пов'язує користувача з його даними. Одна сесія відповідає одному періоду роботи програми (від запуску до закриття).
- Логи (Log Classes): Класи ResourceLog (дані про залізо), AppUsageLog (програми) та InputEvent (клавіатура/миша) пов'язані з Сесією відношенням композиції (або агрегації). Це означає, що лог не може існувати без сесії, до якої він належить.
- Відношення: Реалізовано зв'язок "Один-до-багатьох" (1:N) між User та Session, а також між Session та класами логів.

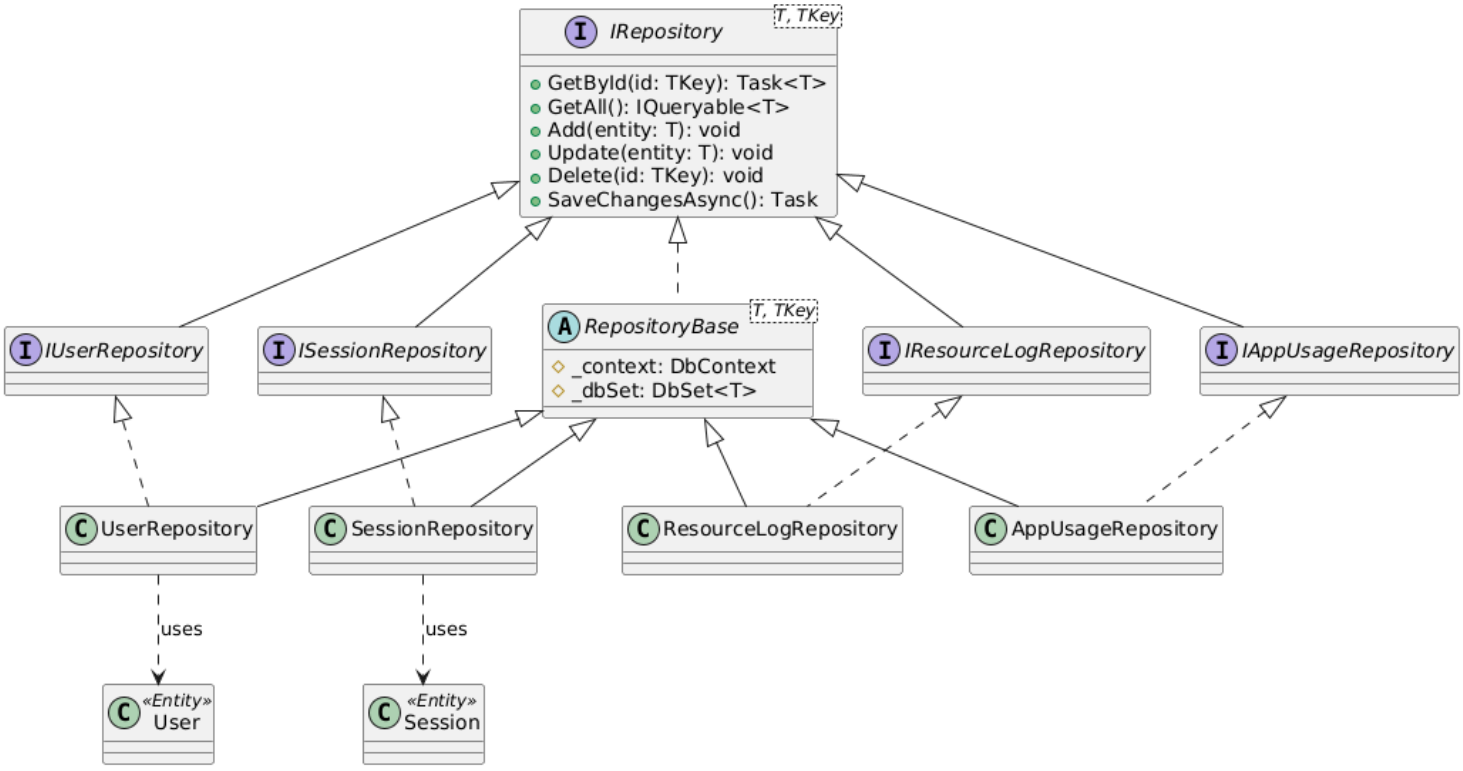

Рис. 4 - Діаграма репозиторіїв

Для абстрагування логіки роботи з базою даних використано шаблон проєктування Repository.

- Generic Interface (IRepository): Визначає контракт для базових CRUD-операцій (Get, Add, Update, Delete), що робить код типізованим та безпечним.

- RepositoryBase: Абстрактний клас, що реалізує загальні методи інтерфейсу IRepository. Це дозволяє зменшити обсяг коду в конкретних репозиторіях, винісши спільну логіку роботи з DbContext в одне місце.
- Специфічні репозиторії: Класи UserRepository, SessionRepository тощо успадковують базову реалізацію та можуть розширювати її специфічними методами (наприклад, пошук сесій за датою).
- Такий підхід забезпечує слабку зв'язаність (Low Coupling) між бізнес-логікою та шаром доступу до даних, що спрощує подальше тестування та масштабування системи.

Програмна реалізація:

BaseEntity.cs:

```
using System;

namespace SystemActivityMonitor.Data.Entities

{

    public abstract class BaseEntity

    {

        public Guid Id { get; set; } = Guid.NewGuid();

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    }

}
```

ResourceLog.cs:

```
using System;

namespace SystemActivityMonitor.Data.Entities

{

    public class ResourceLog : BaseEntity

    {

        public Guid SessionId { get; set; }

        public float CpuLoad { get; set; }

        public float RamUsage { get; set; }

        public Session Session { get; set; }

    }

}
```

Session.cs:

```
using System;

using System.Collections.Generic;

namespace SystemActivityMonitor.Data.Entities

{
```

```
    public class Session : BaseEntity
    {
        public Guid UserId { get; set; }
        public DateTime? EndedAt { get; set; }
        public string MachineName { get; set; }
        public User User { get; set; }
        public ICollection<ResourceLog> ResourceLogs { get; set; } = new List<ResourceLog>();
    }
}
```

User.cs:

```
using System.Collections.Generic;
namespace SystemActivityMonitor.Data.Entities
{
    public class User : BaseEntity
    {
        public string Username { get; set; }
        public string PasswordHash { get; set; }
        public string Role { get; set; }
        public string SettingsJson { get; set; }
        public ICollection<Session> Sessions { get; set; } = new List<Session>();
    }
}
```

Irepository.cs:

```
using System.Linq;
using System.Threading.Tasks;
namespace SystemActivityMonitor.Data.Repositories
{
    public interface IRepository<T, TKey> where T : class
    {
        Task<T> GetById(TKey id);
        IQueryable<T> GetAll();
        void Add(T entity);
        void Update(T entity);
        void Delete(T entity);
        Task SaveChangesAsync();
    }
}
```

RepositoryBase.cs:

```
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
```

```
namespace SystemActivityMonitor.Data.Repositories
{
    public abstract class RepositoryBase<T, TKey> : IRepository<T, TKey> where T : class
    {
        protected readonly MonitorDbContext _context;
        protected readonly DbSet<T> _dbSet;

        public RepositoryBase(MonitorDbContext context)
        {
            _context = context;
            _dbSet = context.Set<T>();
        }

        public async Task<T> GetById(TKey id)
        {
            return await _dbSet.FindAsync(id);
        }

        public IQueryable<T> GetAll()
        {
            return _dbSet.AsQueryable();
        }

        public void Add(T entity)
        {
            _dbSet.Add(entity);
        }

        public void Update(T entity)
        {
            _dbSet.Update(entity);
        }

        public void Delete(T entity)
        {
            _dbSet.Remove(entity);
        }

        public async Task SaveChangesAsync()
        {
            await _context.SaveChangesAsync();
        }
    }
}
```

UserRepository.cs:

```
using System;
using SystemActivityMonitor.Data.Entities;

namespace SystemActivityMonitor.Data.Repositories
{
    public interface IUserRepository : IRepository<User, Guid> { }

    public class UserRepository : RepositoryBase<User, Guid>, IUserRepository
    {
        public UserRepository(MonitorDbContext context) : base(context)
        {
        }
    }
}
```

5. Висновок

У ході виконання лабораторної роботи спроєктовано архітектуру програмної системи "System Activity Monitor". За допомогою мови UML розроблено діаграму варіантів використання для визначення функціональних вимог (автоматичний збір метрик, генерація звітів) та діаграму класів, що відображає статичну структуру предметної області. На практиці створено каркас застосунку мовою C# та реалізовано шар доступу до даних із використанням патерну Repository та технології Entity Framework Core. Це забезпечило слабку зв'язаність компонентів, типізовану роботу з базою даних та можливість легкого масштабування системи. Розроблена структура даних (сутності User, Session, Logs) та архітектура проєкту є готовим фундаментом для подальшої імплементації логіки моніторингу системних ресурсів у наступних етапах роботи.

6. Питання до лабораторної роботи:

1) **Що таке UML?** – це універсальна мова моделювання, яку застосовують для опису та візуального представлення систем.
2) **Що таке діаграма класів UML?** – це графічне відображення класів, їхніх властивостей, методів і взаємозв’язків.
3) **Які UML-діаграми називають канонічними?** – це основні, стандартні діаграми: класів, об’єктів, варіантів використання, послідовностей та інші.
4) **Що таке діаграма варіантів використання?** – вона показує користувачів системи (акторів) та функції, які вони можуть виконувати.
5) **Що таке варіант використання?** – конкретна функція або дія, яку виконує користувач у системі.

6) **Які відношення можна зобразити на діаграмі використання?** – асоціацію, зв’язки include та extend, а також узагальнення.
7) **Що таке сценарій?** – детальний опис послідовності кроків взаємодії між користувачем і системою.
8) **Що таке діаграма класів?** – модель, що демонструє структуру системи через класи та їхні зв’язки.
9) **Які типи зв’язків між класами існують?** – асоціація, агрегація, композиція, наслідування та залежність.
10) **Чим відрізняється композиція від агрегації?** – у композиції частина не може існувати окремо від цілого, тоді як в агрегації частини є більш самостійними.
11) **Чим графічно відрізняється агрегація від композиції на діаграмі класів?** – агрегація позначається порожнім ромбом, композиція — зафарбованим.
12) **Що таке нормальні форми баз даних?** – це набір правил, які допомагають правильно структурувати таблиці та зменшити дублювання даних.
13) **Що таке фізична та логічна моделі БД?** – фізична описує реальне зберігання даних, логічна — концептуальну структуру та взаємозв’язки.
14) **Який зв’язок між таблицями БД і класами програмування?** – зазвичай таблиця відповідає класу, рядок — екземпляру класу, а стовпець — його полю.