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»

«2. HTTP-сервер»

Виконав:
студент групи - ІА-32
Непотачев Іван Дмитрович

Перевірив:
Мягкий Михайло Юрійович

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

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 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 (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.

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.

Діаграми варіантів використання є відправною точкою при збиранні вимог

до програмного продукту та його реалізації. Дана модель будується на аналітичному етапі побудови програмного продукту (збір та аналіз вимог) і дозволяє бізнес-аналітикам отримати більш повне уявлення про необхідне програмне забезпечення та документувати його.

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).

Сценарії використання – це текстові уявлення тих процесів, які відбуваються при взаємодії користувачів системи та самої системи. Вони є чітко формалізованими, покроковими інструкціями, що описують той чи інший процес у термінах кроків досягнення мети. Сценарії використання однозначно визначають кінцевий результат.

Діаграми класів використовуються при моделюванні програмних систем найчастіше. Вони є однією із форм статичного опису системи з погляду її проєктування, показуючи її структуру [3]. 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.

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.

Хід роботи

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Проаналізувати тему та спроєктувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.

• Спроєктувати діаграму класів предметної області.

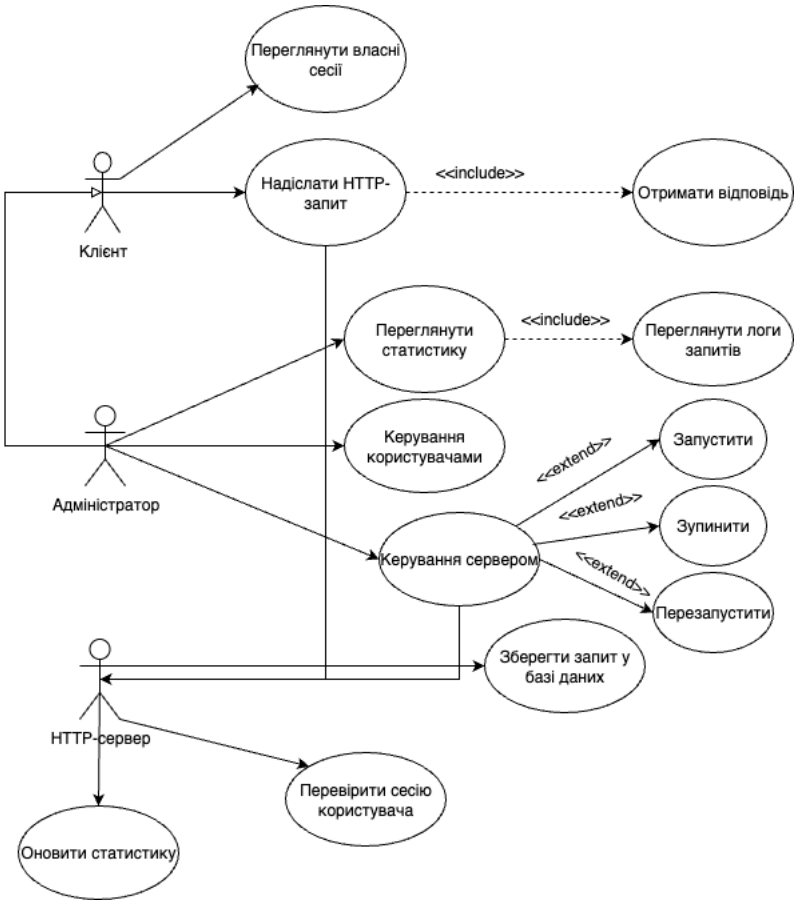
• Вибрати 3 варіанти використання та написати за ними сценарії використання.

• 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.

Діаграма варіантів



Сценарій 1. Надсилання HTTP-запиту:

Передумови: Клієнт має доступ до сервера та підключення до мережі.
Постумови: Клієнт отримує HTTP-відповідь від сервера.
Взаємодіючі сторони: Клієнт, HTTP-сервер.
Короткий опис: Клієнт надсилає HTTP-запит, сервер обробляє його та повертає відповідь.

Основний перебіг подій:

- Клієнт відправляє запит.
- Сервер приймає запит.
- RequestHandler аналізує метод.
- Генерується відповідь (HttpResponse).
- Відповідь надсилається клієнту.

Винятки:

- Виняток №1: Невідомий метод запиту. Сервер повертає повідомлення про помилку (HTTP 405 Method Not Allowed).
- Виняток №2: Внутрішня помилка сервера. Сервер повертає помилку (HTTP 500 Internal Server Error).

Примітки: Відсутні.

Сценарій 2. Керування сервером (запуск, зупинка)
Передумови: Сервер встановлений, адміністратор має доступ до керування.
Постумови: Сервер успішно запущений або зупинений.
Взаємодіючі сторони: Адміністратор, HTTP-сервер.
Короткий опис: Адміністратор може запускати та зупиняти сервер для керування його роботою.

Основний перебіг подій:

- Адміністратор обирає дію («Запустити сервер» або «Зупинити сервер»).
- Якщо обрана дія «Запустити»:
  - Система ініціалізує всі необхідні ресурси.
  - Сервер починає приймати вхідні запити.
- Якщо обрана дія «Зупинити»:
  - Система коректно завершує всі активні з’єднання.
  - Сервер переходить у стан «зупинено».

Винятки:

- Виняток №1: Недостатньо прав доступу. Система відхиляє команду.
- Виняток №2: Порт уже зайнятий іншим процесом при запуску. Сервер не запускається.
- Виняток №3: Сервер не відповідає під час спроби зупинки. Система примусово завершує процес.

Примітки: У випадку помилок адміністратор отримує повідомлення у консолі або логах сервера.

Сценарій 3. Збір статистики
Передумови: Сервер працює, є доступ до бази даних.
Постумови: Запит збережено у статистиці, адміністратор може переглянути його пізніше.
Взаємодіючі сторони: Клієнт, HTTP-сервер, Адміністратор.
Короткий опис: Сервер автоматично зберігає дані про запит у базі даних для подальшого аналізу.

Основний перебіг подій:

- Клієнт надсилає запит.
- RequestHandler передає інформацію про запит у Statistics.
- Statistics формує запис для БД.

- RequestRepository зберігає запис у таблиці Requests.

- Адміністратор у будь-який момент може отримати дані зі статистики.

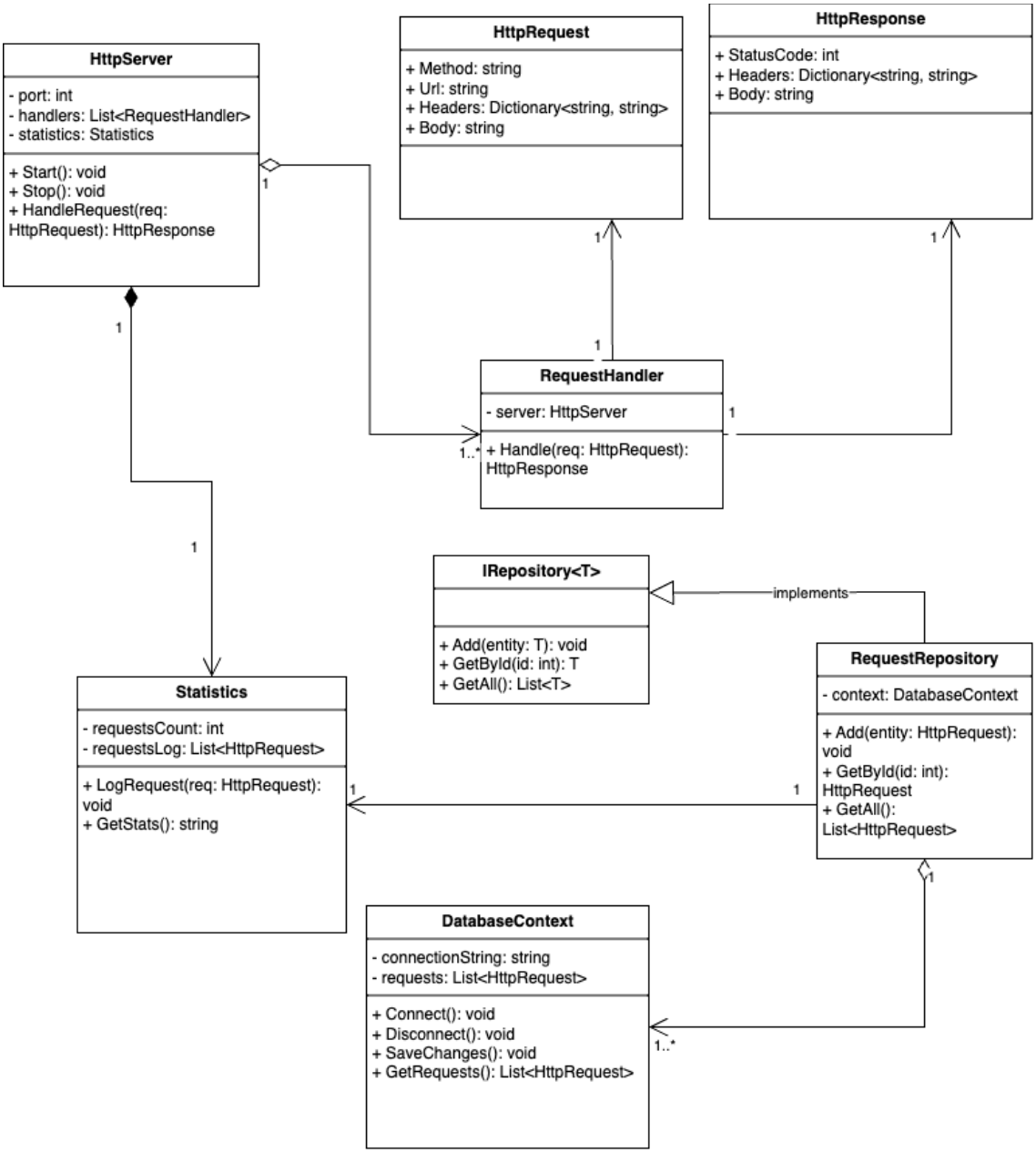
Винятки:

- Виняток №1: Немає з’єднання з базою даних. Запис не зберігається, але сервер все одно обробляє запит.

- Виняток №2: Дані пошкоджені або невалідні. Сервер записує мінімальну інформацію (наприклад, тільки час).

Примітки: Відсутні.

Діаграма класів



1. HttpServer – RequestHandler:

- Тип відносин**:** Агрегація (частина-ціле)
- Пояснення**:** HttpServer містить один або кілька RequestHandler-ів. RequestHandler не існує окремо від сервера.

2. HttpServer – Statistics:

- Тип відносин: Композиція
- Пояснення: Statistics є невід’ємною частиною HttpServer. Без сервера статистика не існує.

3. RequestHandler– HttpRequest / HttpResponse:

- Тип відносин: Асоціація
- Пояснення: RequestHandler отримує HttpRequest та повертає HttpResponse. Тут немає власності або «ціле-частина», просто використання.

4. RequestRepository – IRepository:

- Тип відносин: Узагальнення (implements)
- Пояснення: RequestRepository реалізує інтерфейс IRepository<HttpRequest>.

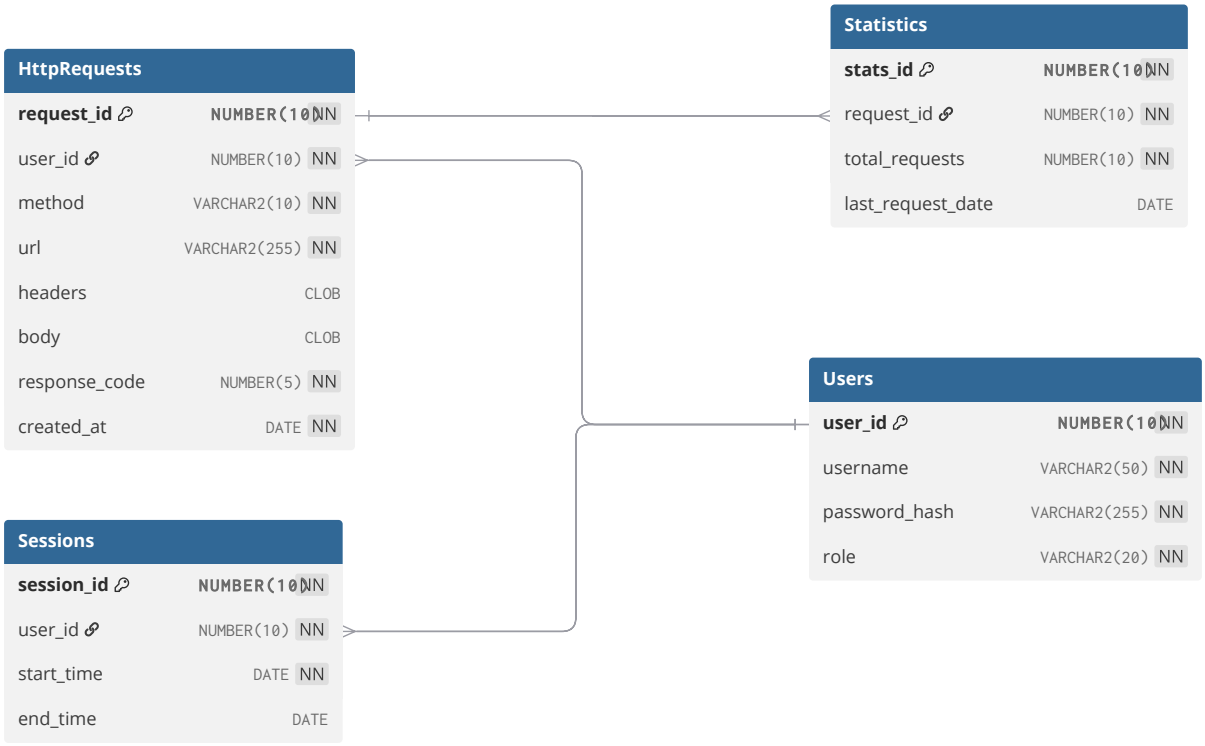
5. RequestRepository – DatabaseContext:

- Тип відносин: Агрегація.
- Пояснення: RequestRepository використовує DatabaseContext для доступу до БД, але контекст може існувати самостійно.

6. Statistics – RequestRepository:

- Тип відносин: Асоціація.
- Пояснення: Statistics викликає методи RequestRepository для збереження та отримання даних.

Проєктування БД



Вихідні коди класів системи

```
public interface IRepository<T>
{
    void Add(T entity);
    T GetById(int id);
    IEnumerable<T> GetAll();
}

public class HttpRequest
{
    public int Id { get; set; }
    public string Method { get; set; }
    public string Url { get; set; }
```

```
    public string Headers { get; set; }
    public string Body { get; set; }
    public int ResponseCode { get; set; }
    public DateTime CreatedAt { get; set; }
}

public class HttpResponse
{
    public int StatusCode { get; set; }
    public string Content { get; set; }
}

public class Statistics
{
    public int Id { get; set; }
    public int TotalRequests { get; set; }
    public DateTime? LastRequestDate { get; set; }
}

public class User
{
    public int Id { get; set; }
    public string Username { get; set; }
    public string PasswordHash { get; set; }
    public string Role { get; set; }
```

```
}

public class DatabaseContext
{
    public List<HttpRequest> Requests { get; set; } = new List<HttpRequest>();
    public List<Statistics> Stats { get; set; } = new List<Statistics>();
    public List<User> Users { get; set; } = new List<User>();

    public void SaveChanges() {}
}

public class RequestRepository : IRepository<HttpRequest>
{
    private readonly DatabaseContext _context;

    public RequestRepository(DatabaseContext context)
    {
        _context = context;
    }

    public void Add(HttpRequest entity)
    {
        _context.Requests.Add(entity);
        _context.SaveChanges();
    }
```

```
    public HttpRequest GetById(int id) =>
        _context.Requests.FirstOrDefault(r => r.Id == id);

    public IEnumerable<HttpRequest> GetAll() =>
        _context.Requests.ToList();
}

public class UserRepository : IRepository<User>
{
    private readonly DatabaseContext _context;

    public UserRepository(DatabaseContext context)
    {
        _context = context;
    }

    public void Add(User entity)
    {
        _context.Users.Add(entity);
        _context.SaveChanges();
    }

    public User GetById(int id) =>
        _context.Users.FirstOrDefault(u => u.Id == id);
```

```
    public IEnumerable<User> GetAll() =>
        _context.Users.ToList();
}

public class StatisticsService
{
    private readonly DatabaseContext _context;

    public StatisticsService(DatabaseContext context)
    {
        _context = context;
    }

    public void UpdateStatistics()
    {
        var stats = _context.Stats.FirstOrDefault();
        if (stats == null)
        {
            stats = new Statistics
            {
                Id = 1,
                TotalRequests = 1,
                LastRequestDate = DateTime.Now
            };
```

```
            _context.Stats.Add(stats);
        }
        else
        {
            stats.TotalRequests++;
            stats.LastRequestDate = DateTime.Now;
        }
        _context.SaveChanges();
    }

    public Statistics GetStatistics() =>
        _context.Stats.FirstOrDefault();
}
```

Висновки

Висновки: під час виконання лабораторної роботи, ми навчилися основам проектування, обрали зручну систему побудови UML-діаграм та навчилися будувати діаграми варіантів використання для системи що проєктується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.