



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»

«НТТР-сервер»

Виконала:

студентка групи - ІА-34

Мартинюк Тетяна

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 проектується, розробляти сценарії варіантів використання та будувати діаграми класів предметної області.

### **Теоретичні відомості**

UML та його діаграми UML (Unified Modeling Language) - уніфікована мова візуального моделювання, що використовується для аналізу, проектування та документування програмних систем. UML дозволяє описувати систему на різних рівнях: від концептуального до фізичного. Основні діаграми UML

- Діаграма варіантів використання (Use Case Diagram) - показує вимоги до системи та взаємодію користувачів із нею.
- Діаграма класів (Class Diagram) - описує статичну структуру системи: класи, їх атрибути, методи та зв'язки.

### **Діаграма варіантів використання**

Діаграма use case відображає функціональність системи з точки зору користувача. Основні елементи:

- Актори (Actor) - користувачі або зовнішні системи.
- Варіанти використання (Use Case) - дії або послуги, які система надає актору (наприклад: вхід, перегляд даних, створення транзакції).

Типи відносин:

- Асоціація - прямий зв'язок актора з варіантом використання.
- Include - один сценарій завжди включає інший (обов'язковий).
- Extend - сценарій може бути розширений додатковим (необов'язковим).
- Узагальнення - спадкування ролей або функціоналу.

Для уточнення роботи системи складаються сценарії використання (use case scenarios), які описують:

- передумови та постумови;
- учасників;
- короткий опис;
- основний перебіг подій;
- винятки.

### **Діаграма класів**

Діаграма класів показує структуру системи: класи, їх атрибути, методи та зв'язки між ними.

Клас містить:

- назву;
- атрибути (дані);
- методи (операції).

Види зв'язків:

- Асоціація - загальний зв'язок між класами.
- Узагальнення (успадкування) - зв'язок між батьківським і дочірнім класом.
- Агрегація - відношення «ціле-частина», де частини можуть існувати окремо.
- Композиція - сильне відношення «ціле-частина», де частини не існують без цілого.

### **Логічна структура бази даних**

Проектування бази даних часто виконується на основі діаграми класів.

Виділяють:

- Фізичну модель - організація файлів і способів зберігання.
- Логічну модель - таблиці, атрибути, ключі, зв'язки.

Щоб уникнути надмірності даних застосовують нормалізацію:

- 1НФ - кожен атрибут має лише одне атомарне значення.
- 2НФ - усі неключові атрибути залежать від усього первинного ключа.
- 3НФ - немає транзитивних залежностей (атрибутів, що залежать від інших неключових атрибутів).
- НФ Бойса-Кодда (BCNF) - посилена форма 3НФ, кожна залежність визначається ключем.

## **Хід роботи**

### **Завдання:**

- Ознайомитись з короткими теоретичними відомостями.
- Проаналізувати тему та спроектувати діаграму варіантів використання відповідно до обраної теми лабораторного циклу.
- Спроектувати діаграму класів предметної області.
- Вибрати 3 варіанти використання та написати за ними сценарії використання.
- На основі спроектованої 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.

### **1. Розробка діаграми варіантів використання**

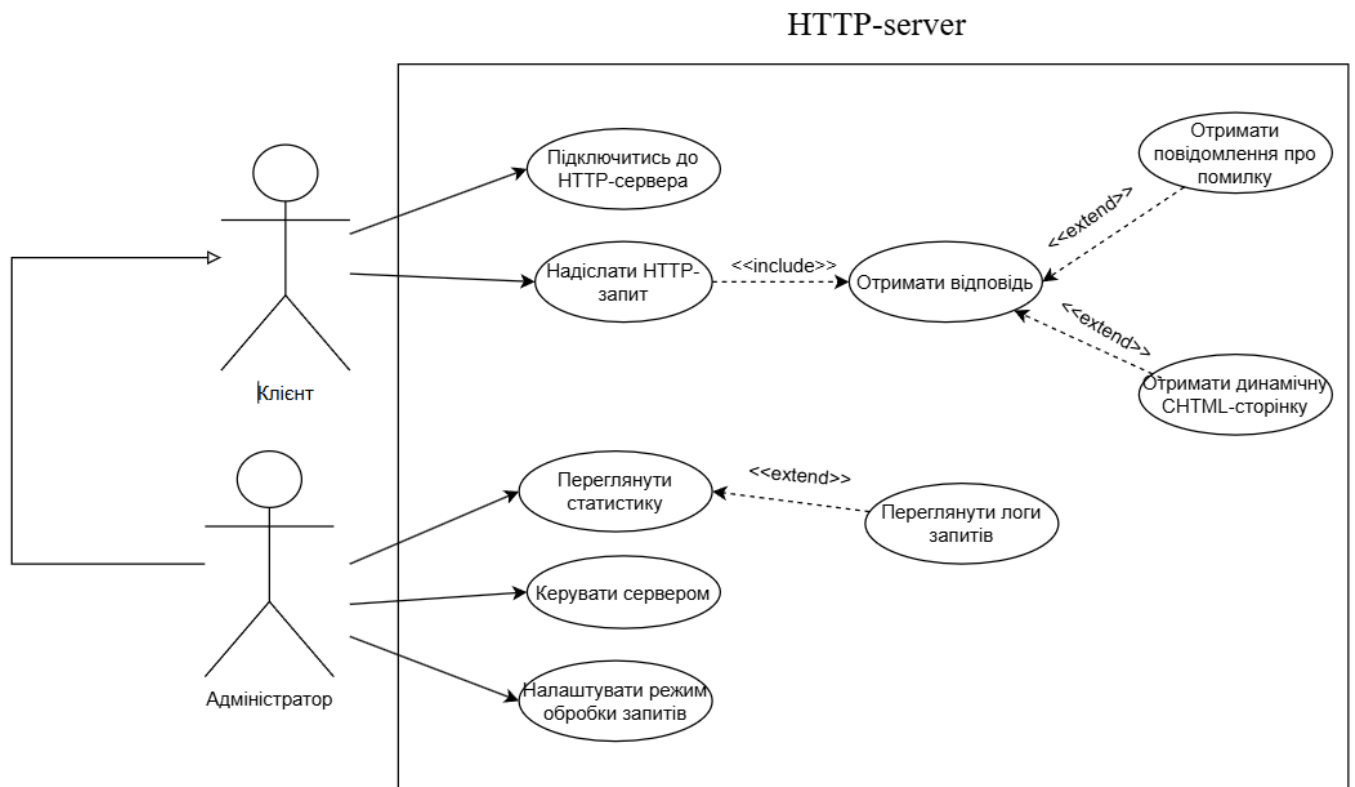


Рисунок 1 – Use case діаграма для користувача та адміністратора

## 2. Сценарії використання

### 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).

**Сценарій 2. Перегляд логів запитів****Передумови:**

Сервер працює, база даних містить записи логів, адміністратор має доступ до інтерфейсу керування.

**Постумови:**

Адміністратор отримує список логів із деталями запитів та відповідей.

**Взаємодіючі сторони:**

Адміністратор, HTTP-сервер, LogRepository.

**Короткий опис:**

Адміністратор переглядає історію запитів, які були оброблені сервером, включаючи статуси, час, помилки та вміст запитів/відповідей.

**Основний перебіг подій:**

- Адміністратор відкриває AdminInterface.
- Вибирає опцію «Переглянути логи».
- Система звертається до LogRepository.
- LogRepository виконує запит до таблиці Logs.
- Для кожного запису система отримує пов'язаний HttpRequest та HttpResponse.
- Адміністратор бачить список логів з такими даними:
  - Час запиту
  - Метод і URL
  - Статус відповіді
  - Успішність (isSuccessful)
  - Повідомлення про помилку (якщо є)

**Винятки:**

- Виняток №1: База даних недоступна. Система повідомляє про помилку з'єднання.
- Виняток №2: Записи логів пошкоджені або не містять повної інформації. Система показує часткові дані з відповідним попередженням.

**Примітки:**

- Перегляд логів обмежений правами доступу або роллю користувача.

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).

### 3. Діаграма класів

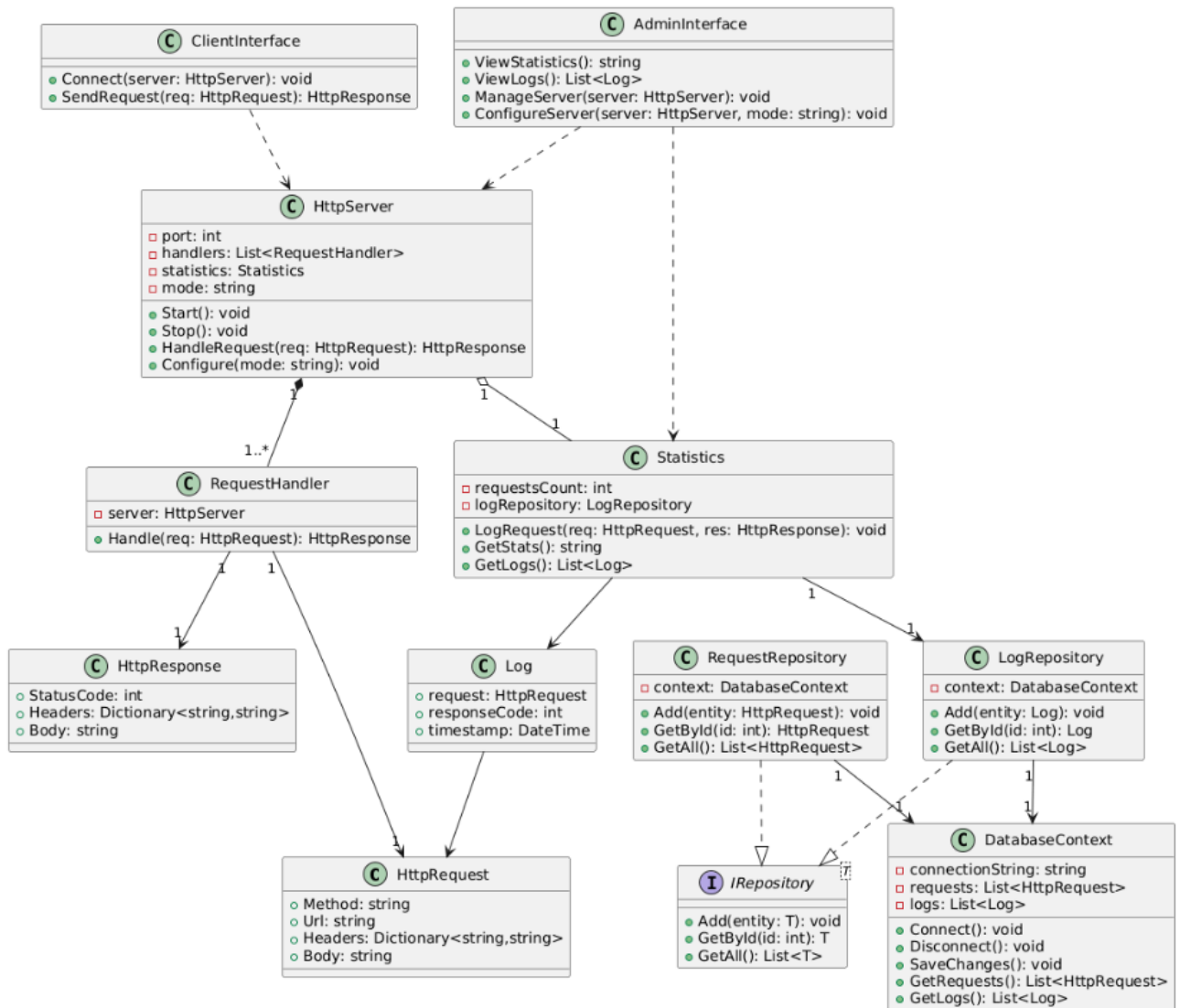


Рисунок 2 – Діаграма класів системи

#### 1. HttpServer – RequestHandler

Тип відносин: Агрегація

Пояснення: HttpServer може мати один або кілька RequestHandler. Вони прив'язані до сервера, але теоретично можуть існувати й незалежно (наприклад, можна підключити до іншого сервера).

#### 2. HttpServer – Statistics

Тип відносин: Композиція

Пояснення: Statistics є невід'ємною частиною сервера. Якщо сервер припиняє роботу, його статистика зникає. Це жорсткий зв'язок «ціле-частина».



### **3. RequestHandler – HttpRequest / HttpResponse**

Тип відносин: Асоціація

Пояснення: RequestHandler працює з вхідними HttpRequest і формує відповідь у вигляді HttpResponse. Це звичайна взаємодія без власності чи ієрархії.

### **4. Statistics – LogRepository**

Тип відносин: Асоціація (використання)

Пояснення: Statistics використовує LogRepository для зберігання та отримання інформації про запити. Це зв'язок "користування сервісом".

### **5. LogRepository – IRepository**

Тип відносин: Реалізація (наслідування від інтерфейсу)

Пояснення: LogRepository реалізує універсальний інтерфейс IRepository<T> для роботи з об'єктами Log. Це дозволяє використовувати патерн Repository.

### **6. RequestRepository – IRepository**

Тип відносин: Реалізація (наслідування від інтерфейсу)

Пояснення: RequestRepository реалізує IRepository<T> для роботи з об'єктами HttpRequest. Завдяки цьому можна централізовано керувати доступом до даних.

### **7. LogRepository – DatabaseContext**

Тип відносин: Асоціація (залежність)

Пояснення: LogRepository використовує DatabaseContext для підключення до БД і збереження логів. Це зв'язок «працює через».

### **8. RequestRepository – DatabaseContext**

Тип відносин: Асоціація (залежність)

Пояснення: `RequestRepository` також працює через `DatabaseContext`, але з даними `HttpRequest`.

## **9. Statistics – Log**

Тип відносин: Асоціація (використання)

Пояснення: `Statistics` створює та аналізує об'єкти `Log`, коли потрібно зберегти дані про виконаний запит.

## **10. Log – HttpRequest**

Тип відносин: Асоціація

Пояснення: `Log` містить інформацію про конкретний `HttpRequest`, який був виконаний.

## **11. AdminInterface – Statistics**

Тип відносин: Асоціація (використання)

Пояснення: Адміністратор через `AdminInterface` може переглядати статистику (`Statistics`). Це інтерфейс для взаємодії.

## **12. AdminInterface – HttpServer**

Тип відносин: Асоціація (керування)

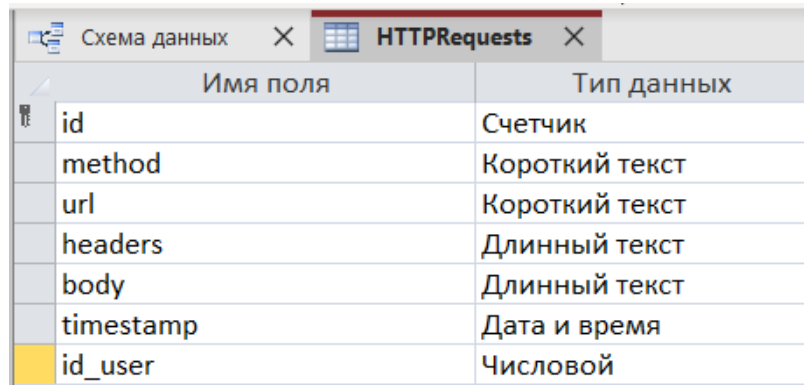
Пояснення: Адміністратор може керувати сервером (`HttpServer`) – запускати, зупиняти або змінювати режим роботи.

## **13. ClientInterface – HttpServer**

Тип відносин: Асоціація (використання)

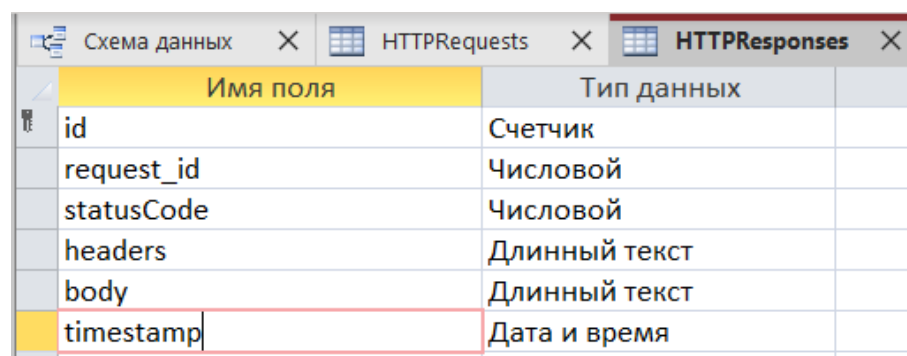
Пояснення: Клієнти підключаються до `HttpServer` через `ClientInterface`, надсилаючи `HttpRequest` і отримуючи `HttpResponse`.

#### 4. Основні класи та структура бази даних системи:



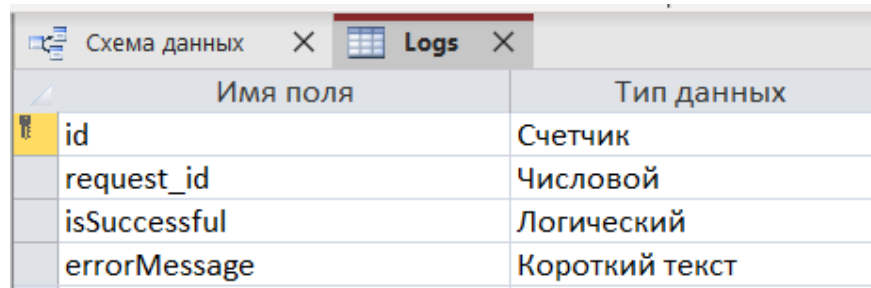
Имя поля	Тип данных
id	Счетчик
method	Короткий текст
url	Короткий текст
headers	Длинный текст
body	Длинный текст
timestamp	Дата и время
id_user	Числовой

Рисунок 3 – Атрибути та їх типи даних для таблиці HTTPRequests



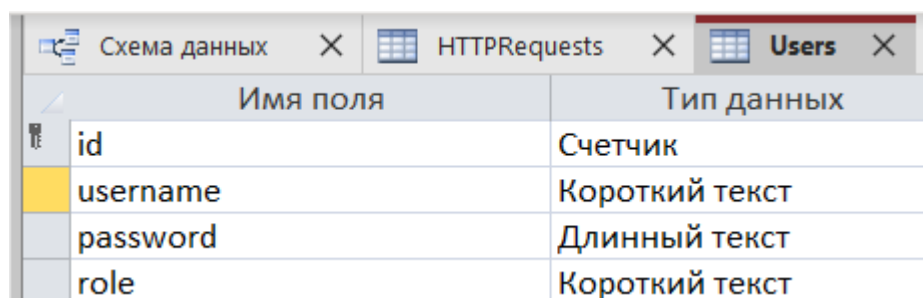
Имя поля	Тип данных
id	Счетчик
request_id	Числовой
statusCode	Числовой
headers	Длинный текст
body	Длинный текст
timestamp	Дата и время

Рисунок 4 – Атрибути та їх типи даних для таблиці HTTPResponses



Имя поля	Тип данных
id	Счетчик
request_id	Числовой
isSuccessful	Логический
errorMessage	Короткий текст

Рисунок 5 – Атрибути та їх типи даних для таблиці Logs



Имя поля	Тип данных
id	Счетчик
username	Короткий текст
password	Длинный текст
role	Короткий текст

Рисунок 6 – Атрибути та їх типи даних для таблиці Users

Имя поля		Тип данных
id		Счетчик
requestsCount		Числовой
timestamp		Дата и время (расширенн
avgResponseTime		Числовой
successesfulRequestsCount		Числовой
errorCount		Числовой

Рисунок 7 – Атрибути та їх типи даних для таблиці Statistics

Важливо помітити, що таблиця Statistics не має зв'язків з іншими таблицями за допомогою вторинних ключів, вона є агрегативною, тобто містить узагальнені показники, сформовані на основі даних з інших таблиць (зокрема Logs, HTTPRequests, HTTPResponses). Її призначення — зберігати попередньо обчислену статистику, яка може бути використана для швидкого доступу до метрик системи без необхідності повторного аналізу сирих даних. Це дозволяє зменшити навантаження на базу даних при частому запиті статистичних звітів.

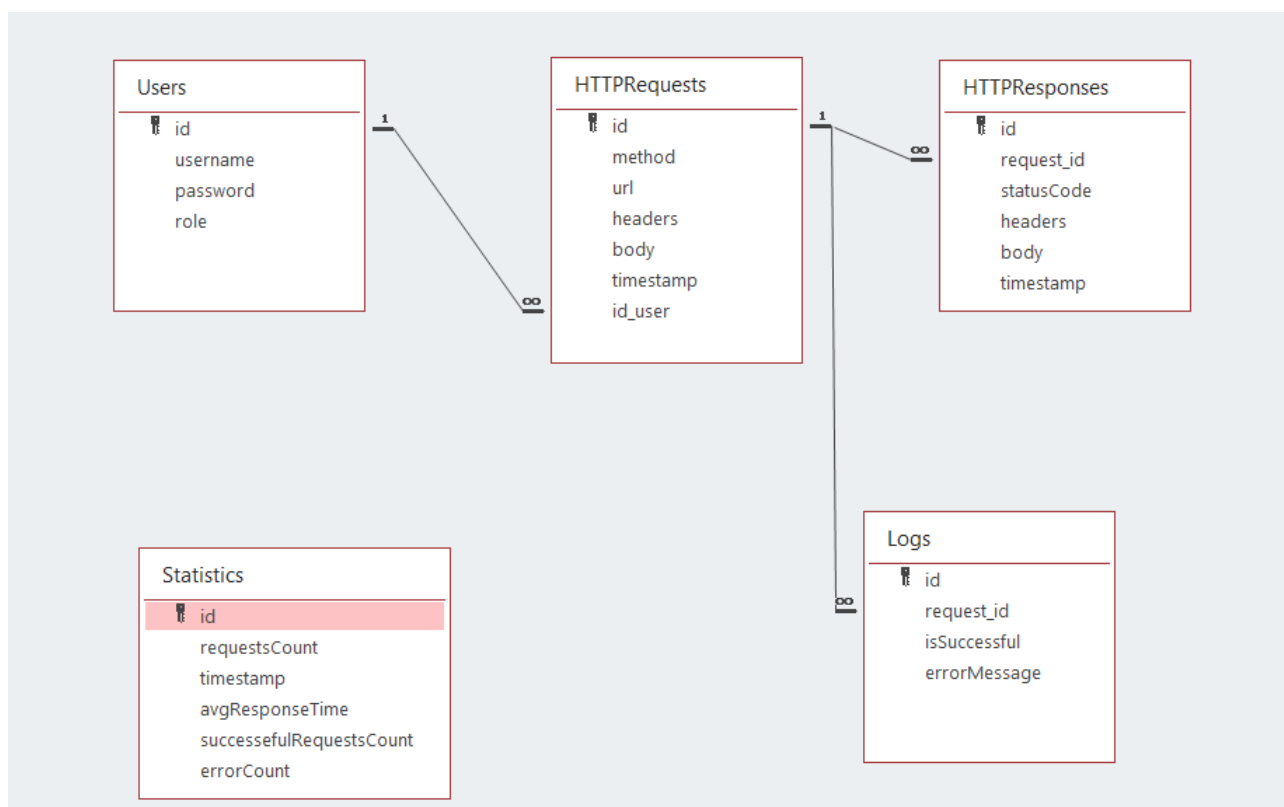


Рисунок 8 – Структура бази даних для предметної області

## 5. Вихідні коди класів системи

```
public interface IRepository<T> {  
    void add(T entity);  
    T getById(int id);  
    List<T> getAll();  
}
```

```
public class HttpRequest {  
    private String method;  
    private String url;  
    private Map<String, String> headers;  
    private String body;  
  
    public String getMethod() { return method; }  
    public void setMethod(String method) { this.method = method; }  
  
    public String getUrl() { return url; }  
    public void setUrl(String url) { this.url = url; }  
  
    public Map<String, String> getHeaders() { return headers; }  
    public void setHeaders(Map<String, String> headers) { this.headers = headers; }  
  
    public String getBody() { return body; }  
    public void setBody(String body) { this.body = body; }  
}  
  
public class HttpResponse {  
    private int statusCode;
```

```
private Map<String, String> headers;
private String body;

public int getStatusCode() { return statusCode; }
public void setStatusCode(int statusCode) { this.statusCode = statusCode; }

public Map<String, String> getHeaders() { return headers; }
public void setHeaders(Map<String, String> headers) { this.headers = headers; }

public String getBody() { return body; }
public void setBody(String body) { this.body = body; }
}

public class Log {
    private HttpRequest request;
    private int responseCode;
    private Date timestamp;

    public Log(HttpRequest request, int responseCode, Date timestamp) {
        this.request = request;
        this.responseCode = responseCode;
        this.timestamp = timestamp;
    }

    public HttpRequest getRequest() { return request; }
    public int getResponseCode() { return responseCode; }
    public Date getTimestamp() { return timestamp; }
}
```

```

public class Statistics {
    private int requestsCount;
    private LogRepository logRepository;

    public Statistics(LogRepository logRepository) {
        this.logRepository = logRepository;
        this.requestsCount = 0;
    }

    public void logRequest(HttpServletRequest req, HttpServletResponse res) {
        requestsCount++;
        logRepository.add(new Log(req, res.getStatusCode(), new Date()));
    }

    public String getStats() {
        return "Total requests: " + requestsCount;
    }

    public List<Log> getLogs() {
        return logRepository.getAll();
    }
}

public class DatabaseContext {
    private String connectionString;
    private List<HttpServletRequest> requests = new ArrayList<>();
    private List<Log> logs = new ArrayList<>();
}

```

```
public void connect()
public void disconnect()
public void saveChanges()
```

```
public List<HttpRequest> getRequests() { return requests; }
public List<Log> getLogs() { return logs; }
}
```

```
public class RequestRepository implements IRepository<HttpRequest> {
    private DatabaseContext context;
```

```
    public RequestRepository(DatabaseContext context) {
        this.context = context;
    }
```

```
@Override
```

```
public void add(HttpRequest entity) {
    context.getRequests().add(entity);
    context.saveChanges();
}
```

```
@Override
```

```
public HttpRequest getById(int id) {
    return context.getRequests().stream().filter(r -> r.hashCode() ==
id).findFirst().orElse(null);
}
```

```
@Override
```



```
public List<HttpRequest> getAll() {  
    return context.getRequests();  
}  
}
```

```
public class LogRepository implements IRepository<Log> {  
    private DatabaseContext context;  
  
    public LogRepository(DatabaseContext context) {  
        this.context = context;  
    }
```

```
@Override
```

```
public void add(Log entity) {  
    context.getLogs().add(entity);  
    context.saveChanges();  
}
```

```
@Override
```

```
public Log getById(int id) {  
    return context.getLogs().stream().filter(l -> l.hashCode() ==  
id).findFirst().orElse(null);  
}
```

```
@Override
```

```
public List<Log> getAll() {  
    return context.getLogs();  
}  
}
```

```

public class RequestHandler {
    private HttpServer server;

    public RequestHandler(HttpServer server) {
        this.server = server;
    }

    public HttpResponse handle(HttpRequest req) {
        return new HttpResponse();
    }
}

public class HttpServer {
    private int port;
    private List<RequestHandler> handlers = new ArrayList<>();
    private Statistics statistics;
    private String mode;

    public void start()
    public void stop()
    public void configure(String mode) { this.mode = mode; }
    public HttpResponse handleRequest(HttpRequest req) { return new
    HttpResponse(); }
}

public class ClientInterface {
    public void connect(HttpServer server)

```

```
    public HttpResponse sendRequest(HttpRequest req) { return new HttpResponse();  
    }  
}
```

```
public class AdminInterface {  
    public String viewStatistics() { return ""; }  
    public List<Log> viewLogs() { return new ArrayList<>(); }  
    public void manageServer(HttpServer server) { /* manage */ }  
    public void configureServer(HttpServer server, String mode) {  
server.configure(mode); }  
}
```

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