



Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 3

з дисципліни «Технології розроблення
програмного забезпечення»

Тема: «Основи проектування розгортання»
«2. HTTP-сервер»

Виконала:
студентка групи - ІА-34
Мартинюк Т.В.

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025

Тема: Основи проектування розгортання.

Мета: Навчитися проектувати діаграми розгортання та компонентів для системи що проектується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Тема проекту: HTTP-сервер (state, builder, factory method, mediator, composite, p2p) Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах.

Теоретичні відомості

Діаграми компонентів — показують модулі (компоненти), їхні залежності і артефакти (.jar, таблиці, html). Використовуються для логічного/фізичного поділу системи.

Діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі. Залежно від способу поділу на модулі розрізняють три види діаграм компонентів:

- логічні;
- фізичні;
- виконувані.

Коли використовують логічне розбиття на компоненти, то у такому разі проєктовану систему віртуально уявляють як набір самостійних, автономних модулів (компонентів), що взаємодіють між собою.

Діаграми розгортання — показують фізичні вузли (devices) і середовища виконання (execution environments), на яких розгортаються артефакти; зв'язки зазначають протоколи (HTTP, SQL/ODBC).

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою. Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад, вебсервер).

Діаграми послідовностей — моделюють часову послідовність повідомлень між акторами/об'єктами (HTTP POST/GET: Browser – Server DB – Browser).

Діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML (Unified Modeling Language), який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу. Вона відображає, як об'єкти обмінюються повідомленнями, показуючи порядок і логіку виконання операцій. Діаграма складається з таких основних елементів:

Актори (Actors): Зазвичай позначаються піктограмами або назвами. Це користувачі чи інші системи, які взаємодіють із системою. Актори можуть бути

Об'єкти або класи: Розміщуються горизонтально на діаграмі. Вони позначаються прямокутниками з іменем об'єкта або класу під прямокутником.

Кожен об'єкт має «життєвий цикл», який представлений вертикальною пунктирною лінією (лінія життя).

Повідомлення: Це лінії зі стрілками, які з'єднують об'єкти. Вони показують передачу повідомлень чи виклик методів. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником) та з пунктирною лінією, що показує повернення результату.

Активності: Вказують періоди, протягом яких об'єкт виконує певну дію. На діаграмі це позначається прямокутником, накладеним на лінію життя.

Контрольні структури: Використовуються для відображення умов, циклів або альтернативних сценаріїв. Наприклад, блоки "alt" (альтернатива) або "loop" (цикл).

Хід роботи

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроектувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.
- Розробити діаграму компонентів для проєктованої системи.
- Розробити діаграму розгортання для проєктованої системи.
- Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.
- На основі спроектованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми.

В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи

з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та відображенням на UI).

- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму розгортання з описом, діаграму компонентів системи з описом, діаграми послідовностей, а також вихідний код системи, який було додано в цій лабораторній роботі.

Діаграма розгортання системи

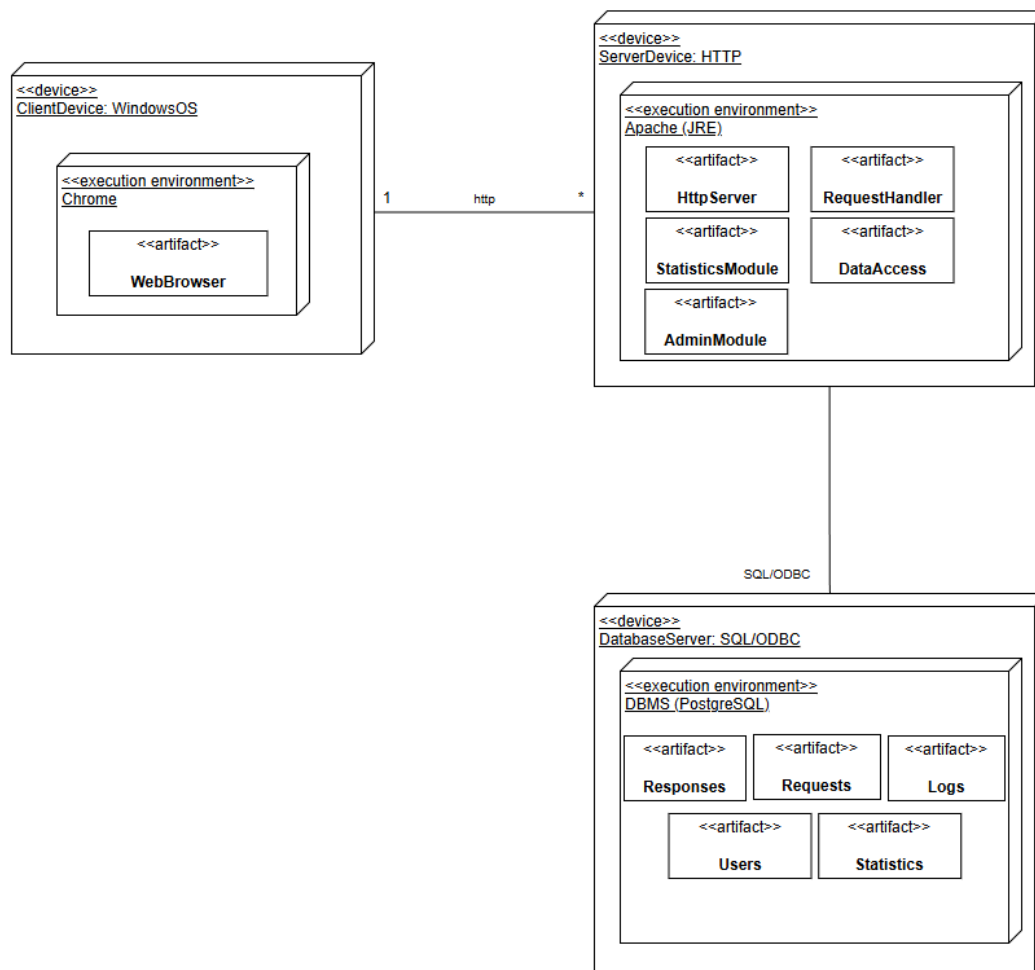


Рисунок 1 - Діаграма розгортання системи

Система складається з трьох ключових фізичних вузлів, які взаємодіють через визначені протоколи:

- **ClientDevice: WindowsOS**: Ноутбук або персональний комп'ютер, що працює під управлінням операційної системи WindowsOS. Це вузол клієнтського рівня.
- **ServerDevice: HTTP**: Серверний пристрій, відповідальний за обробку HTTP-запитів.

- DatabaseServer: SQL/ODBC: Сервер, призначений виключно для зберігання та керування даними.

Кожен вузол містить одне або кілька середовищ виконання та розгорнуті артефакти:

1. ClientDevice

- Середовище виконання: Chrome (<<execution environment>>).
- Артефакт: WebBrowser. Це програмна складова, яка виконує логіку клієнтського інтерфейсу і взаємодіє із сервером.

2. ServerDevice

- Середовище виконання: Java Runtime Environment (JRE) (<<execution environment>>),
- Артефакти:
 - HttpServer та RequestHandler: Ядро сервера та модуль обробки запитів, що виконують основну логіку.
 - StatisticsModule: Модуль, відповідальний за збір, обробку та відображення статистики.
 - DataAccess: Модуль доступу до даних (рівень репозиторіїв), який інкапсулює логіку взаємодії з БД.
 - AdminModule: Модуль керування сервером для адміністратора.

3. DatabaseServer

- Середовище виконання: DBMS (Microsoft SQL Server) (<<execution environment>>). Це система керування базами даних, яка забезпечує зберігання даних.
- Артефакти: Це фізичні сутності даних (таблиці або схеми), що розгорнуті в СУБД: Responses, Requests, Logs, Users та Statistics.

Зв'язки між вузлами вказують на фізичні комунікаційні шляхи:

1. Клієнт - Сервер:

- Протокол: http
- Множинність: Взаємодія відбувається у форматі один клієнт до багатьох серверних процесів (1 *). Сервер може одночасно обслуговувати необмежену кількість клієнтів.

2. Сервер - База Даних:

- Протокол: SQL/ODBC (або інший стандартний протокол доступу до даних).
- Цей зв'язок є необхідним для модуля DataAccess для виконання операцій читання/запису даних.

Діаграма компонентів

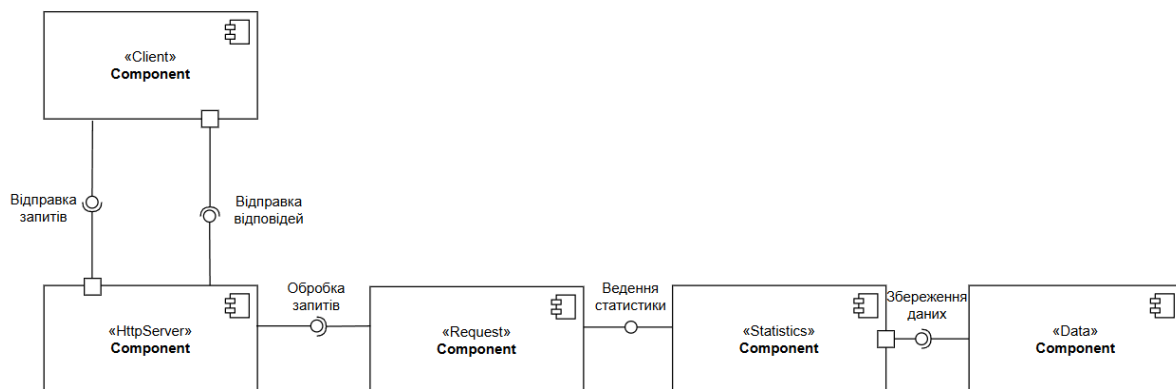


Рисунок 2 - Діаграма компонентів

Основні компоненти системи:

Client — клієнт, який надсилає HTTP-запити до сервера.

Server — головний компонент HTTP-сервера, який отримує запити, отримані від клієнта. Сервер виконує логіку отримання, відправки запитів, відправлення їх на обробку. Також передбачає логіку перенаправлення запитів на інші сервери у разі перевантаження його самого.

Request — цей компонент відповідає за обробку запиту. Його можна розглядати як частину серверної логіки, що включає вибір маршруту для запиту і виклик відповідного обробника. Server передає запити в Request для подальшої обробки.

Statistic — компонент, який відповідає за збір та збереження статистики по запитах. Цей компонент отримує дані з Request і зберігає інформацію для подальшого аналізу.

Data — компонент для роботи з базою даних, де зберігається вся інформація про запити, відповіді та інші дані статистики. Statistic взаємодіє з Data для збереження та отримання статистичних даних.

Діаграма послідовностей

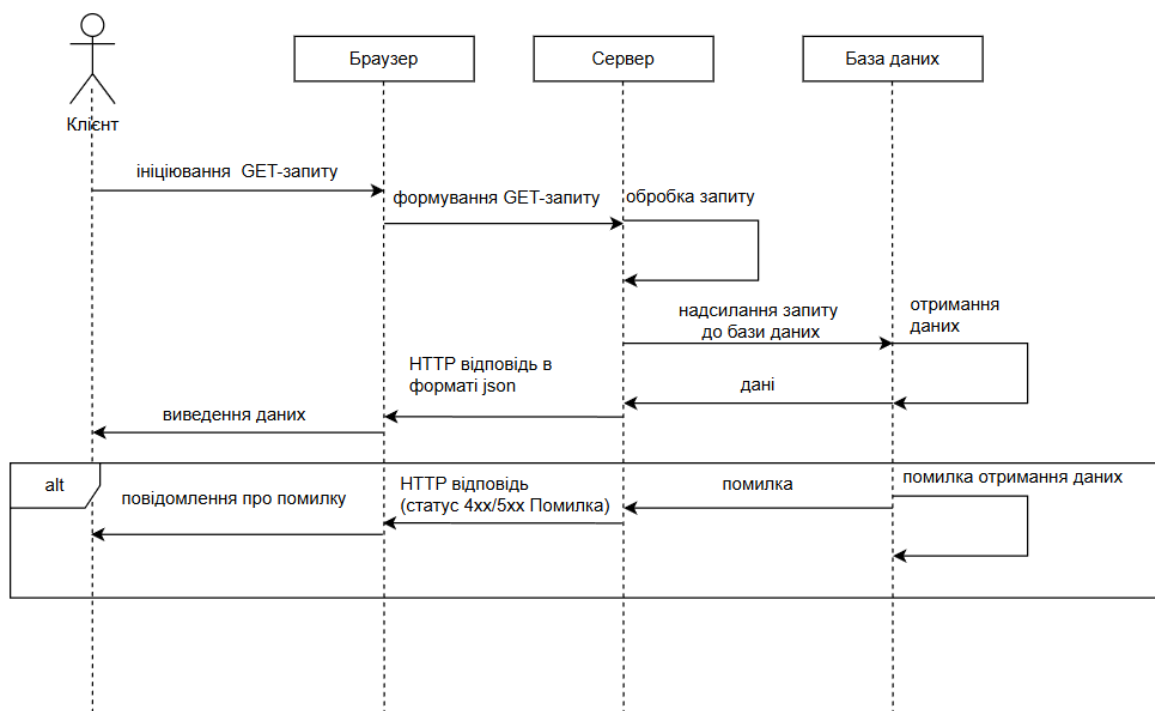


Рисунок 3 – Діаграма послідовностей “Надсилання HTTP GET-запиту”

Прецедент: Надсилання HTTP GET-запиту

Передумови:

1. Клієнт підключений до мережі та має доступ до браузера.

2. Сервер запущений і готовий приймати HTTP-запити.
3. База даних доступна для зчитування даних.

Постумови:

1. Клієнт отримав відповідь від сервера у форматі JSON.
2. У разі помилки – клієнт отримує повідомлення про статус 4xx/5xx.

Взаємодіючі сторони:

1. Клієнт – ініціює запит через браузер.
2. Браузер – формує та надсилає HTTP-запит до сервера.
3. Сервер – обробляє запит і звертається до бази даних.
4. База даних – надає необхідні дані або повідомляє про помилку.

Короткий опис:

Клієнт через браузер надсилає GET-запит на сервер для отримання необхідних даних. Сервер обробляє запит, звертається до бази даних, отримує результати та повертає відповідь у форматі JSON. Якщо під час виконання запиту виникає помилка, сервер формує відповідь із відповідним статусом помилки.

Основний потік подій:

1. Клієнт ініціює GET-запит.
2. Браузер формує HTTP-запит і надсилає його серверу.
3. Сервер приймає запит і розпочинає його обробку.
4. Сервер надсилає запит до бази даних для отримання потрібних даних.
5. База даних повертає дані серверу.
6. Сервер формує HTTP-відповідь у форматі JSON.
7. Браузер приймає відповідь і відображає дані клієнту.

Альтернативний потік:

1. Під час звернення до бази даних виникає помилка.
2. Сервер формує HTTP-відповідь зі статусом 4xx або 5xx
3. Браузер отримує відповідь і відображає повідомлення про помилку клієнту.

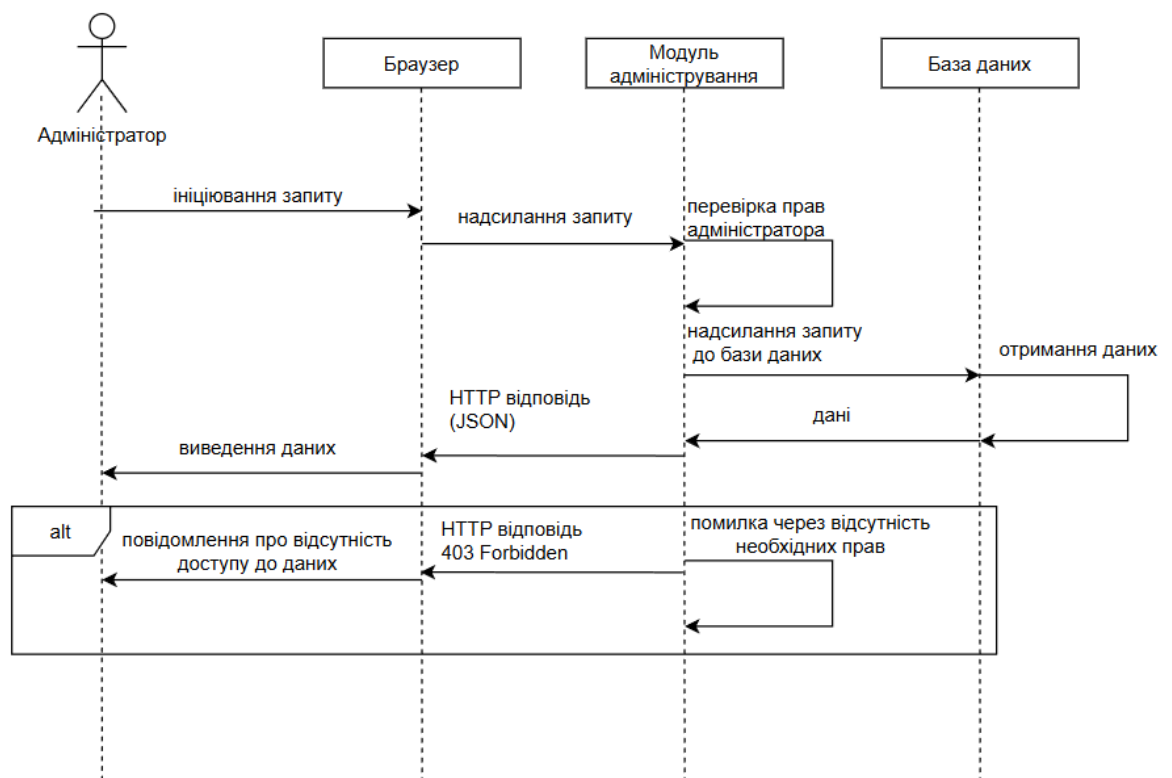


Рисунок 4 - Діаграма послідовності «Отримання статистики адміністратором»

Прецедент: Отримання статистики адміністратором

Передумови:

1. Адміністратор підключений до веб-браузера і має доступ до модуля адміністрування.
2. Сервер працює і збирає статистику запитів.

Постумови:

1. Адміністратор отримав дані статистики у зручному форматі.
2. Сервер надав повідомлення про помилку у разі відсутності прав доступу.

Взаємодіючі сторони:

1. Адміністратор – ініціює запит на отримання статистики.

2. Браузер – передає запит до модуля адміністрування і відображає відповідь.
3. Модуль адміністрування – перевіряє права адміністратора та запитує дані у базі.
4. База даних – надає необхідні дані або повідомляє про помилку.

Короткий опис:

Адміністратор надсилає запит через браузер до модуля адміністрування. Модуль перевіряє права доступу та, у разі наявності необхідних прав, запитує статистику у базі даних і повертає її адміністратору. Якщо прав недостатньо, адміністратор отримує повідомлення про відмову в доступі.

Основний потік подій:

1. Адміністратор ініціює запит на отримання статистики через браузер.
2. Браузер надсилає HTTP-запит до модуля адміністрування.
3. Модуль перевіряє права адміністратора.
4. Якщо права є, модуль надсилає запит до бази даних.
5. База даних надає статистичні дані.
6. Модуль адміністрування формує HTTP-відповідь (JSON) і передає її браузеру.
7. Браузер відображає статистику адміністратору.

Альтернативний потік:

1. Модуль адміністрування визначає, що у користувача недостатньо прав.
2. Модуль формує HTTP-відповідь 403 Forbidden.
3. Браузер відображає повідомлення про відсутність доступу до даних.

Код програми

```
public class WebClient implements ClientInterface {  
    private HttpServer connectedServer;  
  
    @Override
```

```

    public void connect(HttpServer server) {
        this.connectedServer = server;
        System.out.println("Connected to server");
    }

    @Override
    public HttpResponse sendRequest(HttpRequest request) {
        if (connectedServer != null) {
            return connectedServer.handleRequest(request);
        }
        return new HttpResponse(503, "Service Unavailable");
    }
}

public class AdminPanel implements AdminInterface {
    private HttpServer server;
    private Statistics statistics;

    public AdminPanel(HttpServer server, Statistics statistics) {
        this.server = server;
        this.statistics = statistics;
    }

    @Override
    public String viewStatistics() {
        return statistics.getStats();
    }

    @Override
    public List<Log> viewLogs() {
        return statistics.getLogs();
    }

    @Override
    public void manageServer(HttpServer server) {
        server.start();
    }

    @Override
    public void configureServer(HttpServer server, String mode) {
        server.configure(mode);
    }
}

```

```
import java.util.*;
```

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

```
interface ClientInterface {  
    void connect(HttpServer server);  
    HttpResponse sendRequest(HttpRequest request);  
}
```

```
interface AdminInterface {  
    String viewStatistics();  
    List<Log> viewLogs();  
    void manageServer(HttpServer server);  
    void configureServer(HttpServer server, String mode);  
}
```

```
class HttpServer {  
    private int port;  
    private List<RequestHandler> handlers = new ArrayList<>();  
    private Statistics statistics;  
    private String mode;  
  
    public HttpServer(int port, Statistics statistics) {  
        this.port = port;  
        this.statistics = statistics;  
    }  
  
    public void start() {  
        System.out.println("Server started on port " + port);  
    }  
  
    public void stop() {  
        System.out.println("Server stopped.");  
    }  
  
    public HttpResponse handleRequest(HttpRequest request) {
```

```

        for (RequestHandler handler : handlers) {
            HttpResponse response = handler.handle(request);
            statistics.logRequest(request, response);
            return response;
        }
        return new HttpResponse(404, "Not Found");
    }

    public void configure(String mode) {
        this.mode = mode;
        System.out.println("Server mode set to: " + mode);
    }

    public void addHandler(RequestHandler handler) {
        handlers.add(handler);
    }
}

class RequestHandler {
    private HttpServer server;

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

    public HttpResponse handle(HttpRequest request) {
        return new HttpResponse(200, "Request handled successfully");
    }
}

class HttpRequest {
    private String method;
    private String url;
    private Map<String, String> headers = new HashMap<>();
    private String body;

    public HttpRequest(String method, String url, String body) {
        this.method = method;
        this.url = url;
        this.body = body;
    }

    public String getMethod() { return method; }

```

```

    public String getUrl() { return url; }
    public String getBody() { return body; }
}

```

```

class HttpResponse {
    private int statusCode;
    private Map<String, String> headers = new HashMap<>();
    private String body;

    public HttpResponse(int statusCode, String body) {
        this.statusCode = statusCode;
        this.body = body;
    }

    public int getStatusCode() { return statusCode; }
    public String getBody() { return body; }
}

```

```

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

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

    public Date getTimestamp() { return timestamp; }
}

```

```

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

    public void connect() {
        System.out.println("Connected to database");
    }

    public void disconnect() {

```

```

        System.out.println("Disconnected from database");
    }

    public void saveChanges() {
        System.out.println("Database changes saved");
    }

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

class RequestsRepository implements IRepository<HttpRequest> {
    private DatabaseContext context;

    public RequestsRepository(DatabaseContext context) {
        this.context = context;
    }

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

    public HttpRequest getById(int id) {
        return context.getRequests().get(id);
    }

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

class LogRepository implements IRepository<Log> {
    private DatabaseContext context;

    public LogRepository(DatabaseContext context) {
        this.context = context;
    }

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

    public Log getById(int id) {

```

```

        return context.getLogs().get(id);
    }

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

class Statistics {
    private int requestCount;
    private LogRepository logRepository;

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

    public void logRequest(HttpServletRequest request, HttpServletResponse response) {
        logRepository.add(new Log(request, response.getStatusCode()));
        requestCount++;
    }

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

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

public class UsersRepository implements IRepository<User> {
    private DatabaseContext context;

    public UsersRepository(DatabaseContext context) {
        this.context = context;
    }

    @Override
    public void add(User entity) {
        context.getUsers().add(entity);
    }
}

```

```

@Override
public User getById(int id) {
    return context.getUsers().get(id);
}

@Override
public List<User> getAll() {
    return context.getUsers();
}
}

public class User {
    private int id;
    private String username;
    private String role;

    public User(int id, String username, String role) {
        this.id = id;
        this.username = username;
        this.role = role;
    }

    public String getRole() { return role; }
}

public class StatisticsRepository implements IRepository<StatisticsRecord> {
    private DatabaseContext context;

    public StatisticsRepository(DatabaseContext context) {
        this.context = context;
    }

    @Override
    public void add(StatisticsRecord entity) {
        context.getStatistics().add(entity);
    }

    @Override
    public StatisticsRecord getById(int id) {
        return context.getStatistics().get(id);
    }

    @Override
    public List<StatisticsRecord> getAll() {
        return context.getStatistics();
    }
}

```

```

    }
}

public class StatisticsRecord {
    private Date date;
    private int totalRequests;
    private int successfulResponses;
    private int errorResponses;

    public StatisticsRecord(Date date, int totalRequests, int successfulResponses,
int errorResponses) {
        this.date = date;
        this.totalRequests = totalRequests;
        this.successfulResponses = successfulResponses;
        this.errorResponses = errorResponses;
    }
}

```

Висновок: під час виконання лабораторної роботи, ми навчилися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Питання до лабораторної роботи

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) у UML показує фізичне розміщення апаратних вузлів (серверів, клієнтів) і компонентів системи на цих вузлах. Вона відображає, як програмне забезпечення реалізується на апаратних елементах.

2. Які бувають види вузлів на діаграмі розгортання?

Фізичні вузли (Node): реальні апаратні пристрої (сервер, комп'ютер, мобільний пристрій).

Вузли виконання (Execution Environment): середовище, у якому запускаються компоненти (операційна система, віртуальна машина, контейнер).

3. Які бувають зв'язки на діаграмі розгортання?

Асоціація між вузлами (Communication Path): показує можливість обміну даними між вузлами.

Залежність (Dependency): вказує, що один вузол або компонент залежить від іншого.

4. Які елементи присутні на діаграмі компонентів?

- Компоненти (Component): самостійні частини ПЗ.
- Інтерфейси (Interface): порти, через які компоненти взаємодіють.
- Пакети (Package): групування компонентів.
- Зв'язки (Dependency, Association): взаємозв'язки між компонентами.

5. Що становлять собою зв'язки на діаграмі компонентів?

Зв'язки відображають залежності між компонентами: хто на кого покладається, хто надає чи використовує інтерфейс іншого компонента.

6. Які бувають види діаграм взаємодії?

Діаграма послідовностей (Sequence Diagram) – показує порядок повідомлень між об'єктами.

Діаграма комунікацій (Communication Diagram) – показує зв'язки між об'єктами та обмін повідомленнями.

Діаграма часу (Timing Diagram) – показує зміни станів об'єктів у часі.

Діаграма взаємодії (Interaction Overview Diagram) – поєднує елементи кількох діаграм взаємодії.

7. Для чого призначена діаграма послідовностей?

Вона описує порядок і часову послідовність взаємодії між об'єктами системи для реалізації певного сценарію чи варіанту використання.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- Об'єкти/Актори (Lifelines)
- Повідомлення (Messages) – виклики методів між об'єктами
- Активності (Activation bars) – час виконання дії об'єкта
- Події створення/знищення об'єктів

9. Як діаграми послідовностей пов'язані з діаграмами варіантів використання?

Кожна діаграма послідовностей реалізує сценарій певного варіанту використання, деталізуючи, як актори взаємодіють із системою крок за кроком.

10. Як діаграми послідовностей пов'язані з діаграмами класів?

Повідомлення на діаграмі послідовностей зазвичай відповідають методам класів, а об'єкти на діаграмі є екземплярами класів із діаграми класів.