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»

Виконав:
студент групи - ІА-32
Непотачев Іван Дмитрович

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Основи проектування розгортання.

Мета: Навчитися 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.

Тема проєкту: 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.

Зміст

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 спроєктувати 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 спроєктованих 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

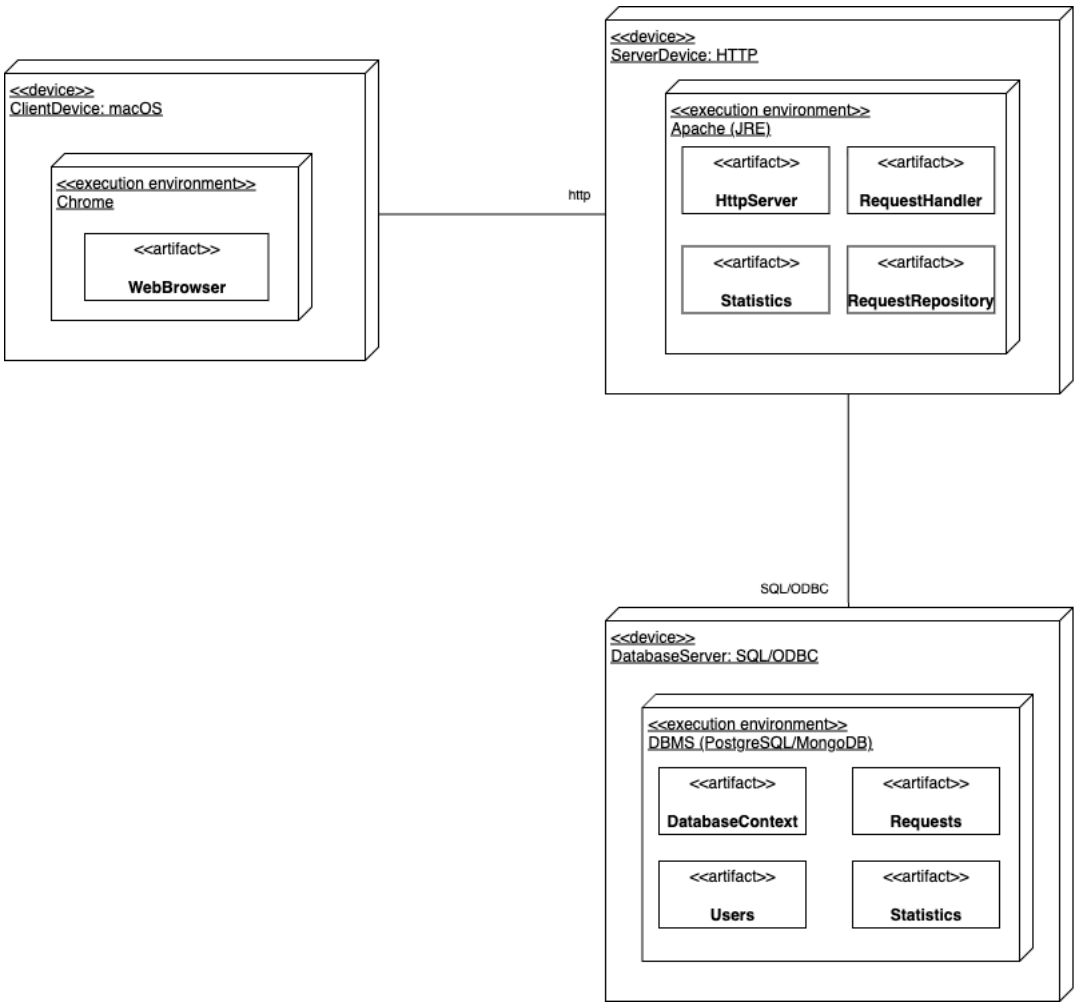

Рис. 1. Діаграма розгортання системи

1. ClientDevice: macOS
   - Це фізичний пристрій користувача (ноутбук чи ПК).
   - Усередині нього розгорнуте середовище виконання Chrome (<<execution environment>>), яке представляє веб-браузер.
   - У браузері запускається артефакт "Web Browser", що відповідає за відображення інтерфейсу користувача і відправку HTTP-запитів на сервер.
2. ServerDevice: HTTP
   - Це серверна машина, що обробляє HTTP-запити.

- Виконується Java Runtime Environment (JRE) (<<execution environment>>), де розміщені серверні артефакти:

- HttpServer.jar – модуль, що приймає HTTP-запити та формує відповіді.

- RequestHandler.jar – обробник запитів від клієнтів.

- Statistics.jar – модуль збору та обробки статистики.

- RequestRepository.jar – модуль доступу до бази даних, який реалізує збереження та вибірку даних.

3. DatabaseServer: SQL/ODBC

- Це сервер бази даних, що відповідає за збереження та обробку даних.

- Усередині нього виконується DBMS (PostgreSQL або MySQL) як середовище виконання.

Розміщені артефакти:

- DatabaseContext – об’єктний шар доступу до БД.

- Requests (table) – таблиця для збереження даних запитів.

- Users (table) – таблиця для збереження інформації про користувачів.

- Statistics (table) – таблиця для накопичення статистичних даних.

4. Зв’язки

- ClientDevice – ServerDevice – взаємодія через HTTP-протокол, коли браузер надсилає запит і отримує відповідь.

- ServerDevice – DatabaseServer – взаємодія через SQL/ODBC, коли сервер звертається до бази даних для збереження чи отримання даних.

Діаграма компонентів

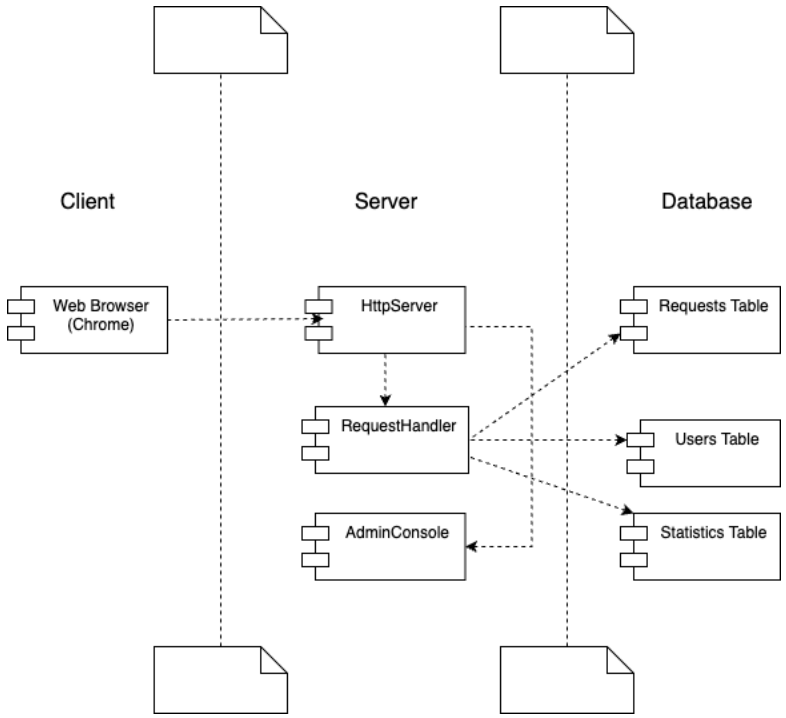

Рис. 2. Діаграма компонентів

1. Client (Клієнтська частина)

   - Компонент Web Browser (Chrome) відповідає за взаємодію з користувачем.

   - Він відправляє HTTP-запити на сервер і відображає відповіді (UI).

2. Server (Серверна частина)

- HttpServer.jar – головний серверний компонент, що приймає HTTP-запити від клієнта та перенаправляє їх на обробку.

- RequestHandler.jar – компонент, який безпосередньо обробляє запити від клієнта:

- AdminConsole.jar – адміністративний інтерфейс сервера. Він взаємодіє з HttpServer.jar для моніторингу та керування сервером.

3. Database (База даних)

- Requests Table – таблиця для збереження інформації про клієнтські запити.

- Users Table – таблиця користувачів, яка зберігає дані для авторизації та профілів.

- Statistics Table – таблиця статистичних даних, що використовується для аналітики та звітності.

4. Зв’язки
   - Web Browser (Chrome) надсилає запити до HttpServer.jar через HTTP.

   - HttpServer.jar перенаправляє запити до RequestHandler.jar.

   - RequestHandler.jar взаємодіє з базою даних

   - AdminConsole.jar взаємодіє з HttpServer.jar для адміністративних функцій.

Діаграма послідовностей

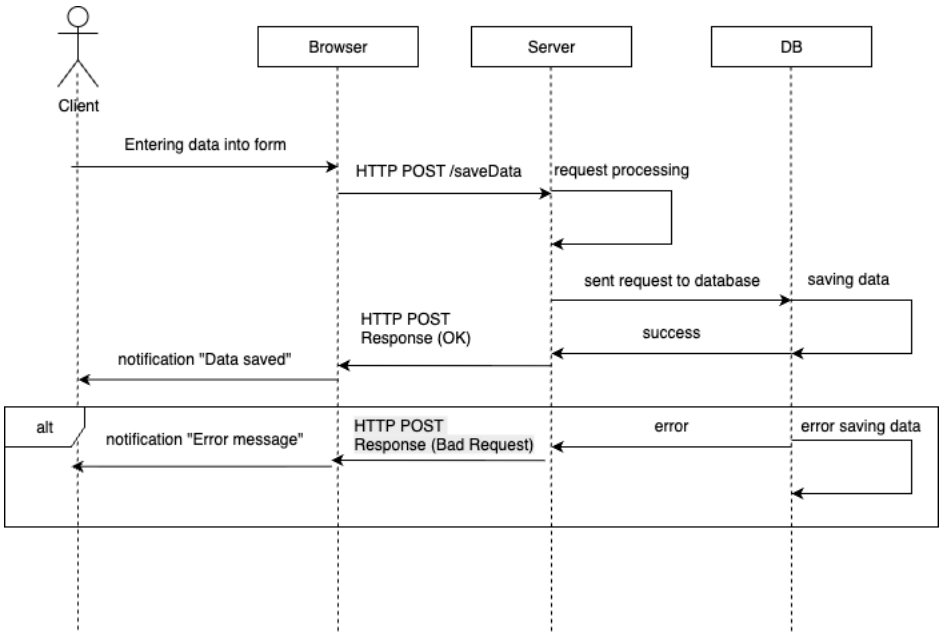

Рис. 3. Діаграма послідовності «Надсилання POST HTTP-запиту»

Перебіг подій:

- Клієнт взаємодіє з браузером.
- Browser відправляє HTTP-запит на Server.
- Server звертається до бази даних і зберігає дані.
- База даних повертає результат.
- Server формує HTTP-відповідь і надсилає у Browser.
- Browser показує результат користувачу.
- У разі винятку, сервер викине помилку і поверне повідомлення з деталями помилки.

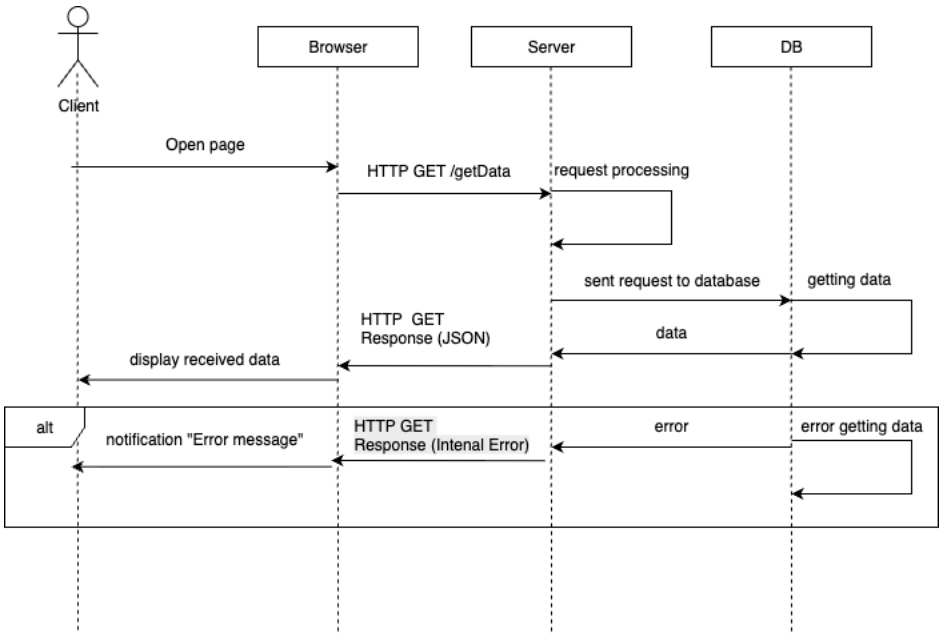

Рис. 4. Діаграма послідовності «Надсилання GET HTTP-запиту»

Перебіг подій:

- Клієнт відкриває сторінку.
- Browser робить HTTP GET-запит на Server.
- Server звертається до бази даних по дані.
- База даних повертає список даних.
- Server відправляє дані назад у Browser.
- Browser відображає їх користувачу.
- У разі винятку, сервер викине помилку і поверне повідомлення з деталями помилки

Код програми

DatabaseContext.java

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class DatabaseContext {
   private final String url;

   public DatabaseContext(String url) {
      this.url = url;
      init();
   }

   private void init() {
      try (Connection conn = getConnection();
           Statement stmt = conn.createStatement()) {
         stmt.executeUpdate("PRAGMA foreign_keys = ON;");
         stmt.executeUpdate(
            "CREATE TABLE IF NOT EXISTS Users (" +
            "user_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "username TEXT UNIQUE NOT NULL, " +
            "password_hash TEXT NOT NULL, " +
            "role TEXT NOT NULL)");
         stmt.executeUpdate(
            "CREATE TABLE IF NOT EXISTS HttpRequests (" +
            "request_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "user_id INTEGER, " +
            "method TEXT NOT NULL, " +
            "url TEXT NOT NULL, " +
            "headers TEXT, " +
            "body TEXT, " +
            "response_code INTEGER NOT NULL, " +
            "created_at TEXT NOT NULL, " +
            "FOREIGN KEY(user_id) REFERENCES Users(user_id) ON DELETE SET
NULL)");
         stmt.executeUpdate(
            "CREATE TABLE IF NOT EXISTS Statistics (" +
            "stats_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
            "total_requests INTEGER NOT NULL, " +
            "last_request_date TEXT)");
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }

   public Connection getConnection() throws SQLException {
      return DriverManager.getConnection(url);
```

```
    }

    // Утиліти для вибірки простих об’єктів
    public List<HttpRequestModel> getAllRequests() {
        String sql = "SELECT request_id, user_id, method, url, headers, body, response_code,
created_at FROM HttpRequests ORDER BY request_id DESC";
        List<HttpRequestModel> list = new ArrayList<>();
        try (Connection c = getConnection();
             PreparedStatement ps = c.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                HttpRequestModel r = new HttpRequestModel();
                r.setId(rs.getInt("request_id"));
                r.setUserId(rs.getObject("user_id") == null ? null : rs.getInt("user_id"));
                r.setMethod(rs.getString("method"));
                r.setUrl(rs.getString("url"));
                r.setHeaders(rs.getString("headers"));
                r.setBody(rs.getString("body"));
                r.setResponseCode(rs.getInt("response_code"));
                r.setCreatedAt(rs.getString("created_at"));
                list.add(r);
            }
        } catch (SQLException ex) { ex.printStackTrace(); }
        return list;
    }
}
```

HttpRequestModel.java

```
public class HttpRequestModel {
    private Integer id;
    private Integer userId;
    private String method;
    private String url;
    private String headers;
    private String body;
    private int responseCode;
    private String createdAt;

    // геттери/сеттери
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public Integer getUserId() { return userId; }
    public void setUserId(Integer userId) { this.userId = userId; }
    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 String getHeaders() { return headers; }
    public void setHeaders(String headers) { this.headers = headers; }
```

```
    public String getBody() { return body; }
    public void setBody(String body) { this.body = body; }
    public int getResponseCode() { return responseCode; }
    public void setResponseCode(int responseCode) { this.responseCode = responseCode; }
    public String getCreatedAt() { return createdAt; }
    public void setCreatedAt(String createdAt) { this.createdAt = createdAt; }
}
```

IRepository.java

```
import java.util.List;

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

RequestRepository.java

```
import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class RequestRepository implements IRepository<HttpRequestModel> {
    private final DatabaseContext ctx;

    public RequestRepository(DatabaseContext ctx) {
        this.ctx = ctx;
    }

    @Override
    public void add(HttpRequestModel entity) throws Exception {
        String sql = "INSERT INTO HttpRequests(user_id, method, url, headers, body,
response_code, created_at) VALUES()";
        try (Connection c = ctx.getConnection();
             PreparedStatement ps = c.prepareStatement(sql)) {
            if (entity.getUserId() == null) ps.setNull(1, Types.INTEGER);
            else ps.setInt(1, entity.getUserId());
            ps.setString(2, entity.getMethod());
            ps.setString(3, entity.getUrl());
            ps.setString(4, entity.getHeaders());
            ps.setString(5, entity.getBody());
            ps.setInt(6, entity.getResponseCode());
            ps.setString(7, LocalDateTime.now().toString());
            ps.executeUpdate();
        }
    }

    @Override
    public HttpRequestModel getById(int id) throws Exception {
```

```
        String sql = "SELECT request_id, user_id, method, url, headers, body, response_code,
created_at FROM HttpRequests WHERE request_id = ?";
        try (Connection c = ctx.getConnection();
             PreparedStatement ps = c.prepareStatement(sql)) {
            ps.setInt(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    HttpRequestModel r = new HttpRequestModel();
                    r.setId(rs.getInt("request_id"));
                    r.setUserId(rs.getObject("user_id") == null ? null : rs.getInt("user_id"));
                    r.setMethod(rs.getString("method"));
                    r.setUrl(rs.getString("url"));
                    r.setHeaders(rs.getString("headers"));
                    r.setBody(rs.getString("body"));
                    r.setResponseCode(rs.getInt("response_code"));
                    r.setCreatedAt(rs.getString("created_at"));
                    return r;
                }
            }
        }
        return null;
    }

    @Override
    public List<HttpRequestModel> getAll() throws Exception {
        return ctx.getAllRequests();
    }
}
```

StatisticsService.java

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class StatisticsService {
    private final DatabaseContext ctx;

    public StatisticsService(DatabaseContext ctx) {
        this.ctx = ctx;
    }

    public void increment() {
        try (Connection c = ctx.getConnection()) {
            // перевіряємо наявність запису
            String select = "SELECT stats_id, total_requests FROM Statistics LIMIT 1";
            try (PreparedStatement ps = c.prepareStatement(select);
                 ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    int id = rs.getInt("stats_id");
```

```
                int total = rs.getInt("total_requests") + 1;
                String update = "UPDATE Statistics SET total_requests = ?, last_request_date =
? WHERE stats_id = ?";
                try (PreparedStatement up = c.prepareStatement(update)) {
                    up.setInt(1, total);
                    up.setString(2, java.time.LocalDateTime.now().toString());
                    up.setInt(3, id);
                    up.executeUpdate();
                }
            } else {
                String insert = "INSERT INTO Statistics(total_requests, last_request_date)
VALUES(?,?)";
                try (PreparedStatement ins = c.prepareStatement(insert)) {
                    ins.setInt(1, 1);
                    ins.setString(2, java.time.LocalDateTime.now().toString());
                    ins.executeUpdate();
                }
            }
        }
    } catch (SQLException ex) { ex.printStackTrace(); }
  }
}
```

SubmitRequestForm.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

public class SubmitRequestForm extends JFrame {
  private final RequestRepository repo;
  private final StatisticsService stats;

  private JTextField txtMethod;
  private JTextField txtUrl;
  private JTextArea txtBody;

  public SubmitRequestForm(RequestRepository repo, StatisticsService stats) {
    this.repo = repo;
    this.stats = stats;
    initUi();
  }

  private void initUi() {
    setTitle("Submit HTTP Request");
    setSize(500, 400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout());

    JPanel top = new JPanel(new GridLayout(2,2));
    top.add(new JLabel("Method:"));
```

```
        txtMethod = new JTextField("GET");
        top.add(txtMethod);
        top.add(new JLabel("URL:"));
        txtUrl = new JTextField("/api/test");
        top.add(txtUrl);

        txtBody = new JTextArea(10, 40);

        JButton btnSend = new JButton("Send (save to DB)");
        btnSend.addActionListener(this::onSend);

        add(top, BorderLayout.NORTH);
        add(new JScrollPane(txtBody), BorderLayout.CENTER);
        add(btnSend, BorderLayout.SOUTH);
    }

    private void onSend(ActionEvent e) {
        try {
            HttpRequestModel r = new HttpRequestModel();
            r.setMethod(txtMethod.getText().trim());
            r.setUrl(txtUrl.getText().trim());
            r.setHeaders(""); // можна додати поле
            r.setBody(txtBody.getText());
            r.setResponseCode(200);
            repo.add(r);                // збережемо в БД
            stats.increment();          // оновимо статистику
            JOptionPane.showMessageDialog(this, "Saved and statistics updated.");
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
        }
    }
}
```

ViewRequestsForm.java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.util.List;

public class ViewRequestsForm extends JFrame {
    private final RequestRepository repo;
    private JTable table;

    public ViewRequestsForm(RequestRepository repo) {
        this.repo = repo;
        initUi();
        loadData();
    }
```

```
    private void initUi() {
        setTitle("Requests Viewer");
        setSize(800, 400);
        setLayout(new BorderLayout());
        table = new JTable();
        add(new JScrollPane(table), BorderLayout.CENTER);
        JButton btnRefresh = new JButton("Refresh");
        btnRefresh.addActionListener(a -> loadData());
        add(btnRefresh, BorderLayout.SOUTH);
    }

    private void loadData() {
        try {
            List<HttpRequestModel> data = repo.getAll();
            String[] cols = {"ID","UserId","Method","URL","Response","CreatedAt"};
            DefaultTableModel m = new DefaultTableModel(cols, 0);
            for (HttpRequestModel r : data) {
                m.addRow(new Object[] {
                    r.getId(),
                    r.getUserId(),
                    r.getMethod(),
                    r.getUrl(),
                    r.getResponseCode(),
                    r.getCreatedAt()
                });
            }
            table.setModel(m);
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error loading data: " + ex.getMessage());
        }
    }
}
```

Main.java

```
import javax.swing.*;

public class Main {
    public static void main(String[] args) {
        String dbUrl = "jdbc:sqlite:server_demo.db";
        DatabaseContext ctx = new DatabaseContext(dbUrl);
        RequestRepository repo = new RequestRepository(ctx);
        StatisticsService stats = new StatisticsService(ctx);

        SwingUtilities.invokeLater(() -> {
            SubmitRequestForm f1 = new SubmitRequestForm(repo, stats);
            f1.setLocation(200, 100);
            f1.setVisible(true);

            ViewRequestsForm f2 = new ViewRequestsForm(repo);
```

```
            f2.setLocation(750, 100);
            f2.setVisible(true);
        });
    }
}
```

Висновки

Висновки: 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?

Ассоціація 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.