Міністерство освіти і науки України
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»

Варіант 11

Виконала:
студентка групи ІА-31
Горлач Д. Д.

Перевірив:
Мягкий М. Ю.

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.

Зміст

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.

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).

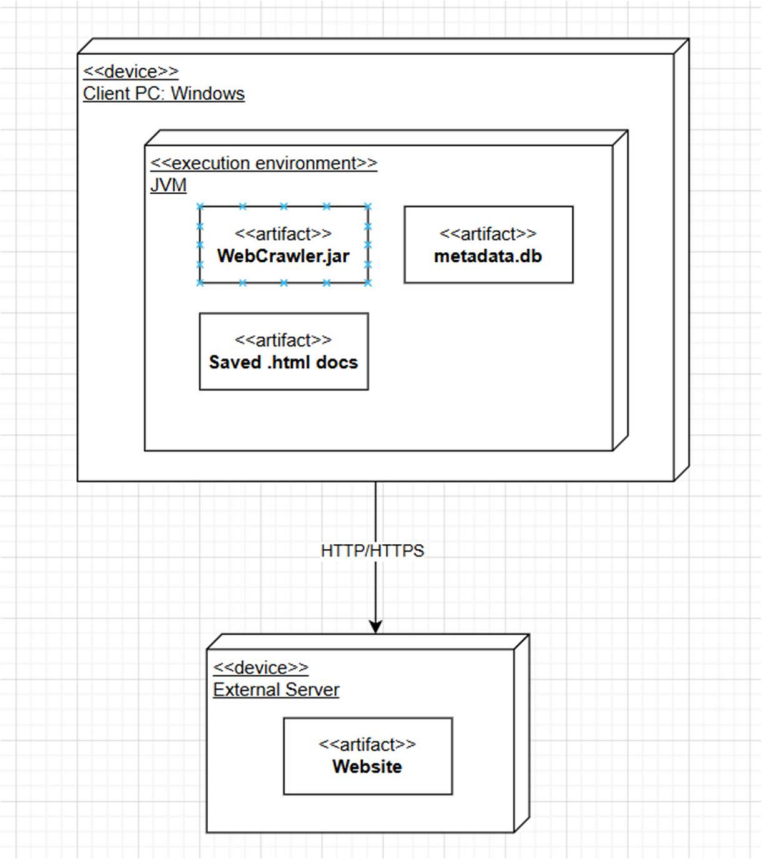
Діаграми послідовностей — моделюють часову послідовність повідомлень між акторами/об’єктами (HTTP POST/GET: Browser – Server DB – Browser).

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:

Діаграма розгортання системи:



1. Клієнтський вузол (Client PC: Windows)

Основним вузлом системи є **Робоча станція користувача** (у діаграмі позначена як Client PC). На відміну від серверних рішень, у даній архітектурі саме цей вузол є центральним, оскільки він розміщує всі компоненти, необхідні для виконання логіки застосунку.

- **Середовище виконання:** Вузол працює під управлінням операційної системи (Windows, macOS або Linux), де розгорнуто віртуальну машину **JVM** (Java Virtual Machine). Саме JVM є стандартизованим середовищем виконання для скомпільованого коду.
- **Виконуваний артефакт:** Головним програмним компонентом є артефакт **WebCrawler.jar**. Цей скомпільований Java-архів містить всю бізнес-логіку сканера: він ініціює HTTP-запити, керує пулом потоків, парсить отриманий HTML, фільтрує контент та керує збереженням даних.
- **Артефакти даних:** WebCrawler.jar під час своєї роботи взаємодіє з двома іншими артефактами, що знаходяться на тому ж вузлі:

- **metadata.db**: Це файл бази даних SQLite, що використовується для зберігання метаданих сканування. Він містить чергу URL-адрес для обходу, список вже відвіданих посилань (для уникнення циклів) та зібрану статистику.
- **Saved .html docs**: Це артефакт, що *створюється* сканером. Він являє собою структурований набір HTML-файлів у файловій системі, що є кінцевим результатом роботи програми.

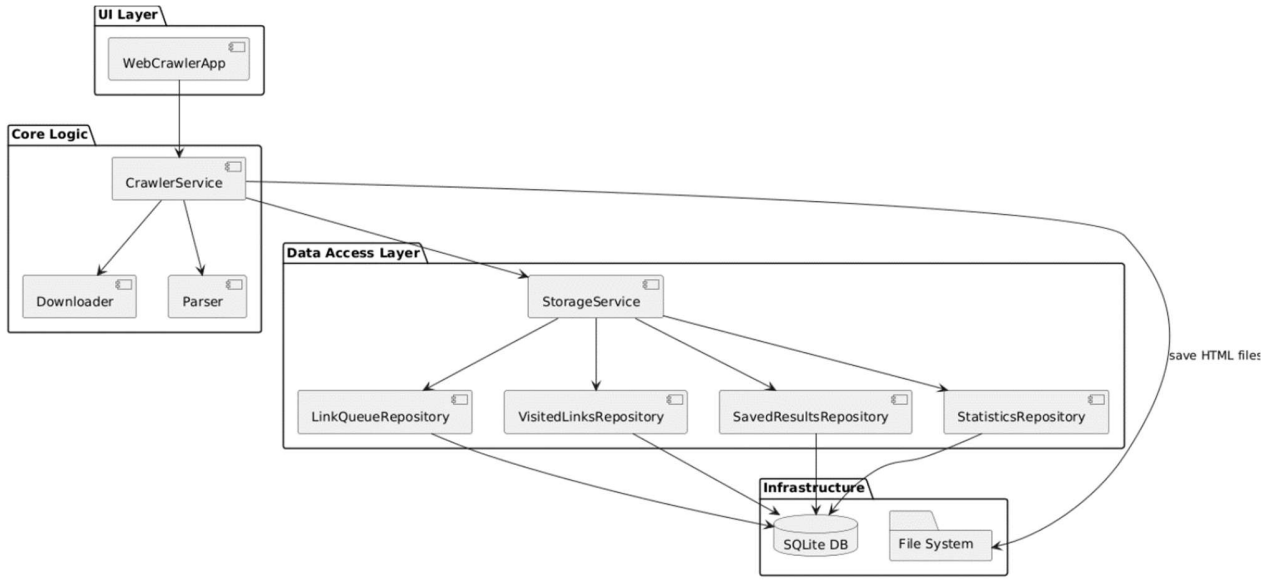
2. Серверний вузол (External Server)

Другим вузлом системи є **Зовнішній сервер**. Це віддалений вузол (або множина вузлів), що знаходиться в Інтернеті та не належить до розроблюваної системи, але є її ціллю.

- **Артефакт:** Цей сервер розміщує артефакт **Website** (веб-сайт), що складається з HTML-сторінок, скриптів, стилів та інших ресурсів.

- **Взаємодія вузлів**
- Зв'язок між вузлами відбувається виключно за ініціативи клієнта. WebCrawler.jar (з вузла Client PC) надсилає **HTTP/HTTPS** запити до Website (на вузлі External Server) для отримання веб-сторінок. Сервер відповідає на ці запити, надсилаючи свій вміст, який клієнтський застосунок потім обробляє.

Діаграма компонентів:



Перебіг подій :

1. Користувач запускає WebCrawlerApp, вводячи початкову адресу та ключове слово.
2. WebCrawlerApp передає керування компоненту CrawlerService, викликаючи метод start(config).
3. CrawlerService формує перший запис у черзі URL через компонент StorageService, який передає запит у:

   LinkQueueRepository → SQLite DB.

4. Починається основний цикл:
   1. CrawlerService через StorageService отримує наступну URL з бази.
   2. StorageService звертається до LinkQueueRepository, який читає дані з SQLite.
   3. CrawlerService перевіряє, чи URL раніше не опрацьована, через:

      VisitedLinksRepository → SQLite DB.

   4. Якщо URL нова:

      CrawlerService викликає Downloader, який завантажує HTML через HTTP.

   5. HTML передається в Parser, який:

- видаляє несемантичні теги,
- шукає ключове слово,

- збирає всі нові посилання.

6. CrawlerService повідомляє StorageService про те, що URL відвідана:VisitedLinksRepository → SQLite DB.
7. Якщо ключове слово знайдено:

CrawlerService зберігає HTML через File System.

StorageService також фіксує дані в SavedResultsRepository → SQLite DB.

8. Parser повертає перелік нових лінків, які додаються в чергу:

LinkQueueRepository → SQLite DB.

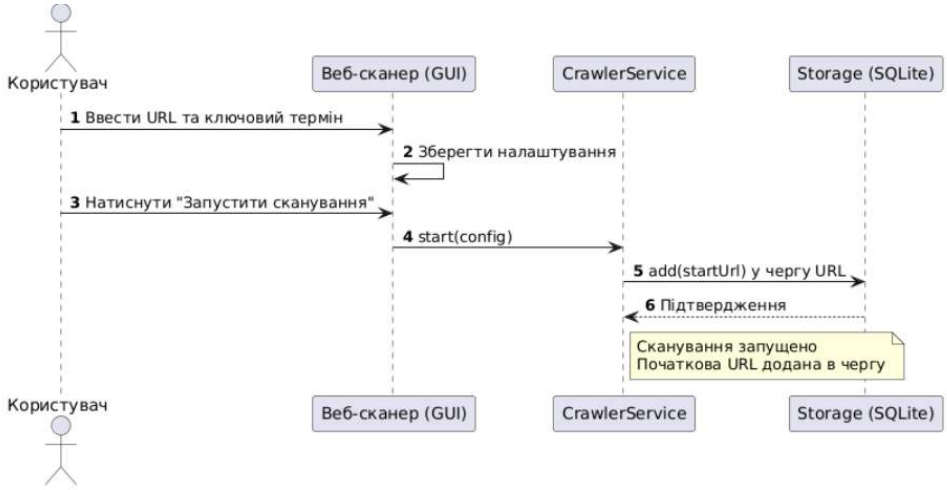
5. Паралельно, за запитом користувача WebCrawlerApp викликає:

showStatistics().тStorageService формує агреговану статистику, звертаючись до:

StatisticsRepository → SQLite DB.

6. WebCrawlerApp відображає статистику користувачу.

Діаграма послідовностей
для сценарію 1(Налаштування та запуск сканування):

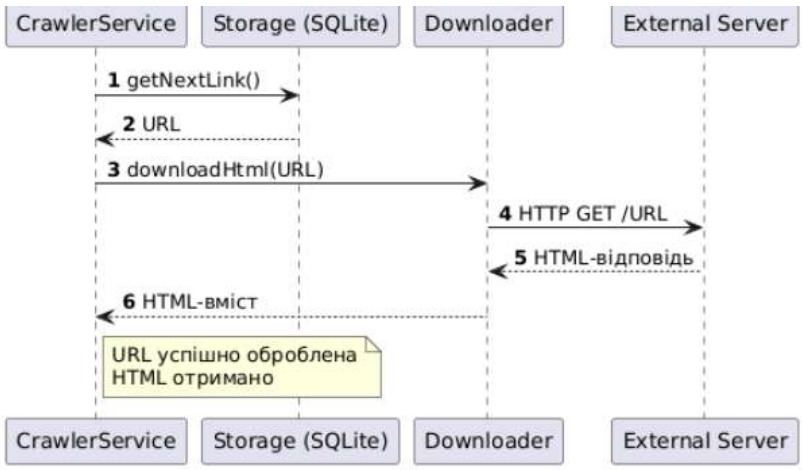


Перебіг подій:

1. Користувач вводить початкову URL-адресу та ключовий термін через GUI застосунку "Веб-сканер".

2. GUI зберігає ці параметри у конфігураційному об’єкті застосунку (config).
3. Користувач натискає кнопку "Запустити сканування".
4. GUI надсилає команду start(config) до CrawlerService для запуску процесу сканування.
5. CrawlerService виконує звернення до Storage (SQLite) через метод add(startUrl) і додає початкову URL у чергу на обхід.
6. Storage повертає підтвердження про додання URL у чергу.
7. Сканування переходить у фоновий режим, готове до обробки черги.

Діаграма послідовностей для сценарію 2 (Обхід сайту та завантаження сторінок):



Перебіг подій:

1. CrawlerService бере наступну URL-адресу з черги через Storage**,** викликавши getNextLink().
2. Storage повертає URL-адресу, яка стоїть першою в черзі.
3. CrawlerService передає URL компоненту Downloader**,** викликавши downloadHtml(URL).
4. Downloader формує HTTP GET-запит і надсилає його на External Server **(**цільовий сервер).
5. External Server обробляє запит і повертає відповідь — HTML-код сторінки (та за потреби CSS, ресурси).
6. Downloader приймає відповідь і передає чистий HTML назад у CrawlerService**.**

7. CrawlerService отримує HTML-вміст і готовий передати його на парсинг або подальшу обробку.

Код програми:

```
package org.example;

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

public class CrawlerSetupForm extends JFrame {

    private JTextField urlField;
    private JTextField keywordField;
    private JButton startButton;

    private WebCrawlerApp app;

    public CrawlerSetupForm(WebCrawlerApp app) {
        this.app = app;

        setTitle("Налаштування сканування");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);

        JLabel urlLabel = new JLabel("Start URL:");
        urlLabel.setBounds(20, 20, 100, 25);
        add(urlLabel);

        urlField = new JTextField();
        urlField.setBounds(120, 20, 200, 25);
        add(urlField);

        JLabel keywordLabel = new JLabel("Keyword:");
```

```
        keywordLabel.setBounds(20, 60, 100, 25);
        add(keywordLabel);

        keywordField = new JTextField();
        keywordField.setBounds(120, 60, 200, 25);
        add(keywordField);

        startButton = new JButton("Start Crawling");
        startButton.setBounds(120, 100, 150, 30);
        add(startButton);

        startButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                app.startCrawling(urlField.getText(), keywordField.getText());
            }
        });
    }
}
```

```
package org.example;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;

public class CrawlerStatsForm extends JFrame {

    private JTable statsTable;
    private WebCrawlerApp app;

    public CrawlerStatsForm(WebCrawlerApp app) {
        this.app = app;

        setTitle("Статистика");
        setSize(500, 300);
```

```
        setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);

        statsTable = new JTable();
        add(new JScrollPane(statsTable));

        refreshStatistics();
    }

    public void refreshStatistics() {
        StatisticsData stats = app.getStatistics();

        DefaultTableModel model = new DefaultTableModel();
        model.addColumn("Visited Pages");
        model.addColumn("Saved Files");
        model.addColumn("Queue Size");

        model.addRow(new Object[]{
                stats.visitedCount,
                stats.savedFiles,
                stats.queueSize
        });

        statsTable.setModel(model);
    }
}



package org.example;

import javax.swing.*;

public class WebCrawlerApp {

    private CrawlerService crawlerService;
    private StorageService storageService;

    private CrawlerSetupForm setupForm;
```

```
    private CrawlerStatsForm statsForm;

    public WebCrawlerApp() {
        storageService = new StorageService();
        crawlerService = new CrawlerService(storageService);

        setupForm = new CrawlerSetupForm(this);
        statsForm = new CrawlerStatsForm(this);

        setupForm.setVisible(true);
    }

    public void startCrawling(String url, String keyword) {
        if (url.isEmpty() || keyword.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Усі поля повинні бути
заповнені!");
            return;
        }

        CrawlConfig config = new CrawlConfig(url, keyword);
        crawlerService.start(config);
        storageService.addToQueue(url);

        JOptionPane.showMessageDialog(null, "Сканування
розпочато!");

        statsForm.refreshStatistics();
        statsForm.setVisible(true);
    }

    public StatisticsData getStatistics() {
        return storageService.getStatistics();
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(WebCrawlerApp::new);
    }
}
```

```
package org.example;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class CrawlerService {

    private StorageService storage;
    private ExecutorService executor;

    public CrawlerService(StorageService storage) {
        this.storage = storage;
        this.executor = Executors.newFixedThreadPool(2);
    }

    public void start(CrawlConfig config) {
        executor.submit(() -> crawl(config.startUrl, config.keyword));
    }

    private void crawl(String url, String keyword) {
        while (true) {
            String nextUrl = storage.getNextFromQueue();
            if (nextUrl == null) break;

            if (storage.isVisited(nextUrl))
                continue;

            storage.markVisited(nextUrl);

            String html = Downloader.download(nextUrl);
            if (html == null) continue;

            boolean contains = Parser.containsKeyword(html, keyword);

            if (contains) {
```

```
            storage.saveResultFile(nextUrl, html);
        }

        for (String link : Parser.extractLinks(html)) {
            storage.addToQueue(link);
        }
    }
  }
}
```

```
package org.example;

import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

public class StorageService {

    private Queue<String> queue = new LinkedList<>();
    private Set<String> visited = new HashSet<>();
    private int savedFiles = 0;

    public void addToQueue(String url) {
        if (!queue.contains(url) && !visited.contains(url))
            queue.add(url);
    }

    public String getNextFromQueue() {
        return queue.poll();
    }

    public boolean isVisited(String url) {
        return visited.contains(url);
    }

    public void markVisited(String url) {
```

```
        visited.add(url);
    }

    public void saveResultFile(String url, String html) {
        try {
            FileWriter writer = new FileWriter(url.replaceAll("[^a-zA-Z0-
9]", "_") + ".html");
            writer.write(html);
            writer.close();
            savedFiles++;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public StatisticsData getStatistics() {
        return new StatisticsData(visited.size(), savedFiles, queue.size());
    }
}
```

```
package org.example;

import java.io.IOException;
import org.jsoup.Jsoup;

public class Downloader {

    public static String download(String url) {
        try {
            return Jsoup.connect(url)
                    .timeout(5000)
                    .ignoreContentType(true)
                    .get()
                    .html();
        } catch (IOException e) {
```

```
            return null;
        }
    }
}
```

```
package org.example;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.select.Elements;
import java.util.ArrayList;
import java.util.List;

public class Parser {

    public static boolean containsKeyword(String html, String keyword) {
        return html.toLowerCase().contains(keyword.toLowerCase());
    }

    public static List<String> extractLinks(String html) {
        List<String> links = new ArrayList<>();

        Document doc = Jsoup.parse(html);
        Elements elements = doc.select("a[href]");

        elements.forEach(e -> links.add(e.absUrl("href")));

        return links;
    }
}
```

```
package org.example;

public class CrawlConfig {
```

```
    public String startUrl;
    public String keyword;

    public CrawlConfig(String startUrl, String keyword) {
        this.startUrl = startUrl;
        this.keyword = keyword;
    }
}
```

```
package org.example;

public class StatisticsData {

    public int visitedCount;
    public int savedFiles;
    public int queueSize;

    public StatisticsData(int visited, int saved, int queue) {
        this.visitedCount = visited;
        this.savedFiles = saved;
        this.queueSize = queue;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>WebCrawlerApp</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>

    <dependencies>

        <!-- JSoup — HTML Parser -->
        <dependency>
            <groupId>org.jsoup</groupId>
            <artifactId>jsoup</artifactId>
            <version>1.17.2</version>
        </dependency>

        <!-- SQLite JDBC Driver -->
        <dependency>
            <groupId>org.xerial</groupId>
            <artifactId>sqlite-jdbc</artifactId>
            <version>3.45.1.0</version>
        </dependency>

    </dependencies>

</project>
```

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.