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»

«Web-browser»

Виконала:
студент групи - ІА-32
Самойленко С. Д.

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: Web-browser

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

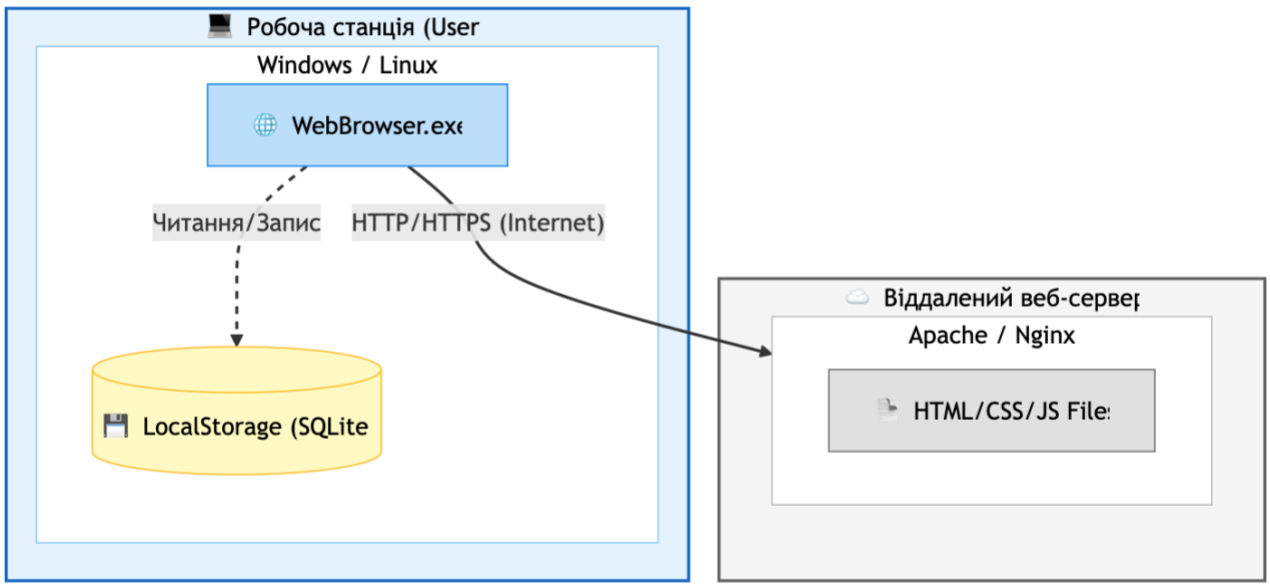

Рисунок 1 - Діаграма розгортання системи

Система складається з двох ключових фізичних вузлів:

1. **User Workstation (Робоча станція користувача):** Фізичний пристрій (ноутбук або ПК), на якому працює користувач.
    - **Середовище:** Операційна система (Windows, macOS або Linux).
    - **Артефакт WebBrowser.exe:** Виконуваний файл нашого браузера.
    - **Артефакт LocalStorage:** Локальна база даних (наприклад, файл SQLite), де браузер зберігає історію, закладки та кеш. На відміну від серверних додатків, браузер зберігає персональні дані локально.
2. **Remote Web Server (Віддалений веб-сервер):** Будь-який сервер в Інтернеті (наприклад, server Google або KPI), до якого звертається браузер.
    - **Артефакт:** Веб-контент (HTML, зображення).

Зв'язок: Вузол користувача зв'язується з сервером через протокол **HTTP/HTTPS**.

Діаграма компонентів

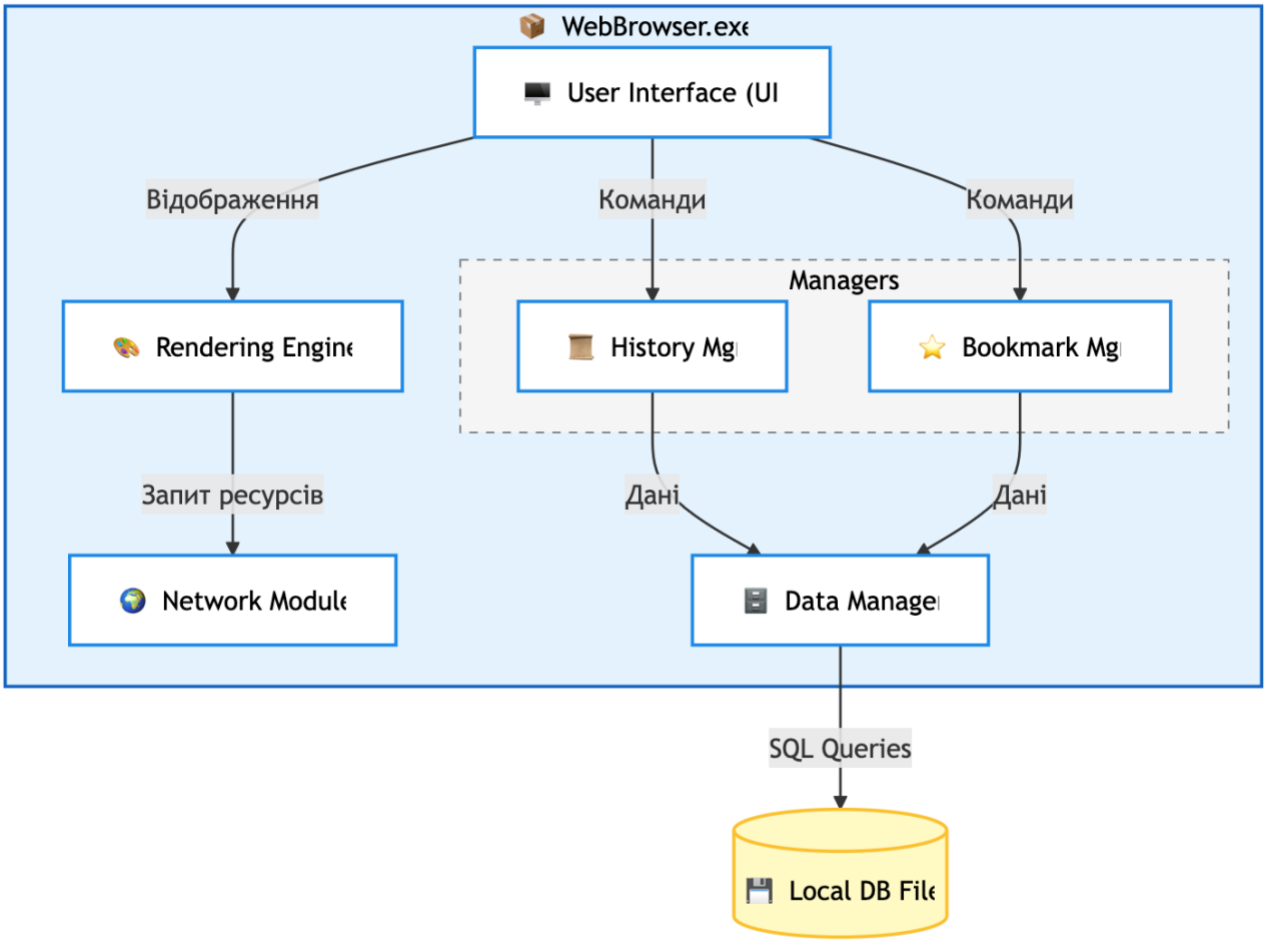

Рисунок 2 - Діаграма компонентів

Опис компонентів:

- **User Interface (UI):** Відповідає за кнопки, адресний рядок і вкладки.
- **Rendering Engine:** "Серце" браузера, яке перетворює HTML-код на візуальне зображення.
- **Network Module:** Відповідає за надсилання HTTP-запитів та отримання відповідей.
- **Data Manager:** Компонент (Repository), що інкапсулює роботу з базою даних.

- **History/Bookmark Managers:** Логічні модулі керування відповідними сутностями.

Діаграма послідовностей

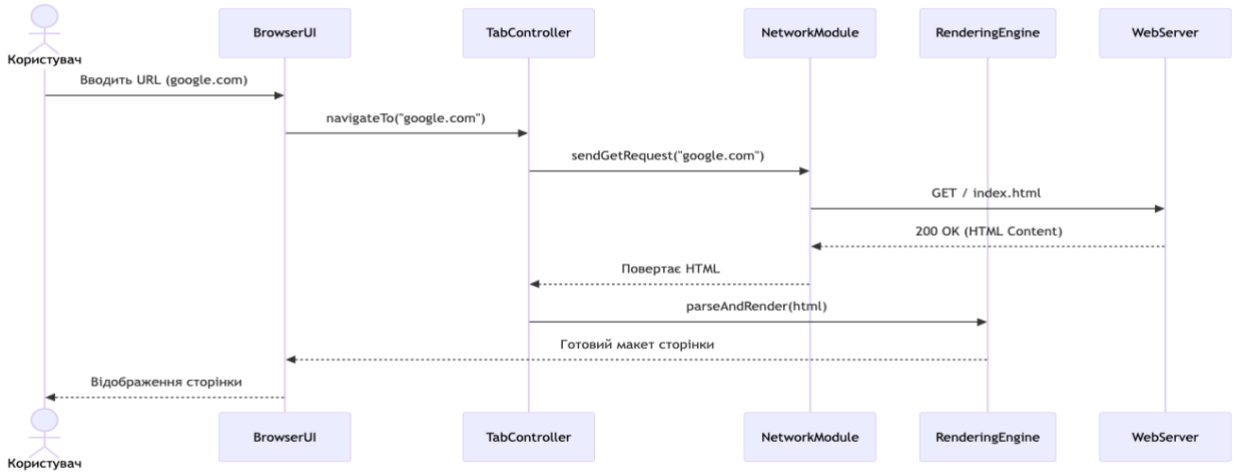

Рисунок 3 – Діаграма послідовностей “Надсилання HTTP GET-запиту”

Прецедент: Відкриття веб-сторінки (GET-запит)

Передумови:

1. Браузер запущений на пристрої клієнта.
2. Користувач має активне підключення до мережі Інтернет.

Постумови:

1. Веб-сторінка успішно відображена у вікні браузера.
2. URL-адреса збережена в історії відвідувань.
3. У разі помилки – клієнт отримує повідомлення про статус 4xx/5xx.

Взаємодіючі сторони:

1. Користувач – ініціює запит через інтерфейс браузера (вводить URL).
2. BrowserUI – приймає ввід та відображає результати.
3. NetworkModule – формує та надсилає HTTP-запит до сервера.
4. WebServer – обробляє запит і повертає контент сторінки.

Короткий опис: Користувач через браузер надсилає GET-запит на сервер для отримання веб-сторінки. Браузер формує запит, звертається до веб-сервера, отримує HTML-код сторінки, обробляє його та відображає візуальний інтерфейс користувачу. Якщо під час виконання запиту виникає помилка, браузер відображає сторінку з відповідним статусом помилки.

Основний потік подій:

1. Користувач вводить URL-адресу та ініціює перехід.
2. BrowserUI передає команду контролеру вкладки.
3. NetworkModule формує HTTP GET-запит і надсилає його серверу.
4. WebServer приймає запит і шукає запитаний ресурс (index.html).
5. WebServer повертає відповідь (код 200 OK) та HTML-контент.
6. NetworkModule передає дані рушію відображення.
7. RenderingEngine будує макет сторінки.
8. BrowserUI відображає готову сторінку клієнту.

Альтернативний потік:

1. Сервер недоступний або ресурс не знайдено.
2. WebServer повертає HTTP-відповідь зі статусом 4xx або 5xx.
3. Браузер отримує відповідь і відображає користувачу повідомлення про помилку завантаження.

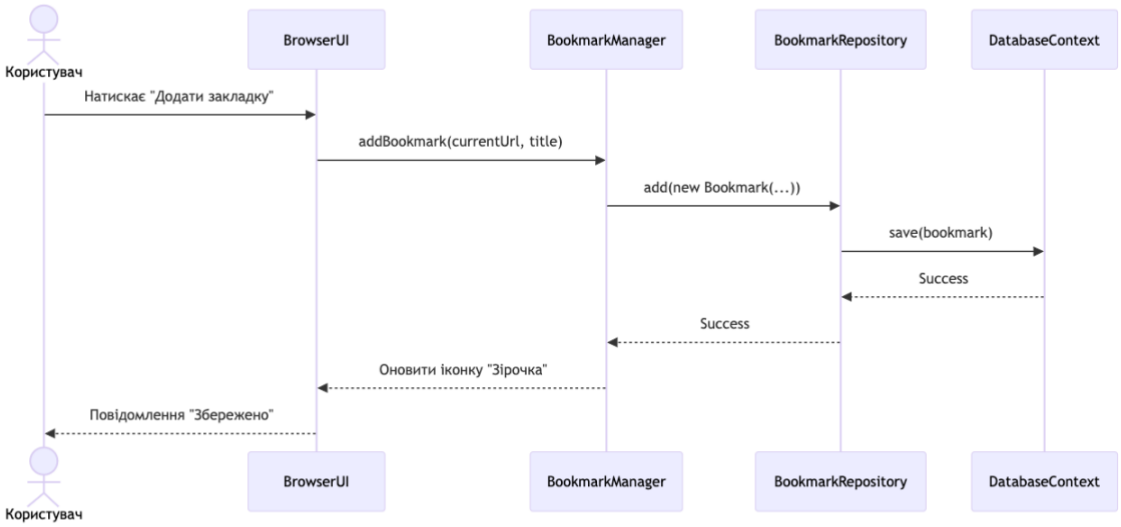

Рисунок 4 - Діаграма послідовності «Отримання статистики адміністратором»

Прецедент: Додавання сторінки до закладок

Передумови:

1. Браузер запущений і в ньому відкрита веб-сторінка.
2. Ця сторінка ще не додана до списку закладок.

Постумови:

1. Дані про сторінку (URL та заголовок) збережено у локальній базі даних.
2. Інтерфейс браузера оновлено (іконка закладки стала активною).
3. У разі помилки – користувач отримує повідомлення про неможливість збереження.

Взаємодіючі сторони:

1. Користувач – ініціює дію натисканням кнопки інтерфейсу.
2. BrowserUI – приймає команду та відображає статус виконання.
3. BookmarkManager – обробляє логіку створення закладки.
4. LocalStorage – фізично зберігає дані на диску.

Короткий опис: Користувач бажає зберегти поточну веб-сторінку для швидкого доступу в майбутньому. Він натискає кнопку «Додати в улюблене» в інтерфейсі. Система зчитує поточну адресу та назву сторінки, формує об'єкт закладки та зберігає його у локальній базі даних. Після успішного запису інтерфейс візуально підтверджує дію (зміна кольору іконки).

Основний потік подій:

1. Користувач натискає кнопку «Додати в улюблене» на панелі інструментів.
2. BrowserUI передає поточний URL та заголовок сторінки до BookmarkManager.
3. BookmarkManager створює новий об'єкт закладки.
4. BookmarkManager звертається до репозиторію для збереження даних у LocalStorage.
5. LocalStorage підтверджує успішний запис даних.
6. BookmarkManager повертає статус успіху до BrowserUI.
7. BrowserUI оновлює іконку закладки та виводить повідомлення «Збережено».

Альтернативний потік:

1. Під час запису у локальне сховище виникає помилка (наприклад, помилка доступу до файлу).
2. LocalStorage повертає помилку.
3. BrowserUI відображає користувачу спливаюче вікно: «Не вдалося зберегти закладку».

Код програми

```
import java.util.*;

// --- Basic Interfaces & Entities (from Lab 2) ---
interface IRepository<T> {
    void add(T entity);
    List<T> getAll();
}

class Bookmark {
    String url;
    String title;
    public Bookmark(String url, String title) { this.url = url; this.title = title; }
    @Override public String toString() { return title + " (" + url + ")"; }
}
```

```
class HistoryItem {
    String url;
    Date date;
    public HistoryItem(String url) { this.url = url; this.date = new Date(); }
    @Override public String toString() { return date + ": " + url; }
}

// --- Data Access Layer (Component: Data Manager) ---
class DatabaseContext {
    // Імітація локальної БД (наприклад, SQLite файл)
    private List<Bookmark> bookmarks = new ArrayList<>();
    private List<HistoryItem> history = new ArrayList<>();

    public List<Bookmark> getBookmarks() { return bookmarks; }
    public List<HistoryItem> getHistory() { return history; }

    public void saveChanges() {
        System.out.println("[DB] Changes saved to local storage file.");
    }
}

class BookmarkRepository implements IRepository<Bookmark> {
    private DatabaseContext db;
    public BookmarkRepository(DatabaseContext db) { this.db = db; }
    public void add(Bookmark entity) { db.getBookmarks().add(entity);
db.saveChanges(); }
    public List<Bookmark> getAll() { return db.getBookmarks(); }
}

class HistoryRepository implements IRepository<HistoryItem> {
    private DatabaseContext db;
    public HistoryRepository(DatabaseContext db) { this.db = db; }
    public void add(HistoryItem entity) { db.getHistory().add(entity);
db.saveChanges(); }
    public List<HistoryItem> getAll() { return db.getHistory(); }
}

// --- Logic Layer (Components: Network, Render, Managers) ---

class NetworkModule {
    public String sendGetRequest(String url) {
        System.out.println("[Network] Sending GET request to " + url + "...");
        // Імітація затримки мережі
        try { Thread.sleep(500); } catch (InterruptedException e) {}

        if (url.contains("google")) return "<html>Google Home Page</html>";
        if (url.contains("kpi")) return "<html>KPI University</html>";
        return "<html>404 Not Found</html>";
    }
}

class RenderingEngine {
    public void render(String html) {
        System.out.println("[Render] Parsing HTML...");
        System.out.println("[Render] Displaying content: " + html);
```

```
    }
}

class Tab {
    private String currentUrl;
    private String title;
    private RenderingEngine renderer;
    private NetworkModule network;
    private HistoryRepository historyRepo;

    public Tab(HistoryRepository hRepo) {
        this.renderer = new RenderingEngine();
        this.network = new NetworkModule();
        this.historyRepo = hRepo;
    }

    public void navigate(String url) {
        this.currentUrl = url;
        // 1. Зберегти в історію
        historyRepo.add(new HistoryItem(url));
        // 2. Завантажити
        String html = network.sendGetRequest(url);
        // 3. Відобразити
        renderer.render(html);
        this.title = "Page: " + url;
    }

    public String getUrl() { return currentUrl; }
    public String getTitle() { return title; }
}

// --- UI Layer (Component: User Interface) ---
class BrowserUI {
    private BookmarkRepository bookmarkRepo;
    private HistoryRepository historyRepo;
    private Tab activeTab;

    public BrowserUI() {
        DatabaseContext db = new DatabaseContext();
        this.bookmarkRepo = new BookmarkRepository(db);
        this.historyRepo = new HistoryRepository(db);
        this.activeTab = new Tab(historyRepo); // Створення вкладки
        System.out.println("[UI] Browser Interface Initialized.");
    }

    public void userEntersUrl(String url) {
        System.out.println("\n>>> User enters URL: " + url);
        activeTab.navigate(url);
    }

    public void userClicksBookmark() {
        System.out.println("\n>>> User clicks 'Add Bookmark'");
        if (activeTab.getUrl() != null) {
            Bookmark b = new Bookmark(activeTab.getUrl(), activeTab.getTitle());
            bookmarkRepo.add(b);
```

```
            System.out.println("[UI] Bookmark added successfully.");
        } else {
            System.out.println("[UI] No page loaded to bookmark.");
        }
    }

    public void userOpensHistory() {
        System.out.println("\n>>> User opens History");
        for (HistoryItem item : historyRepo.getAll()) {
            System.out.println(item);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        BrowserUI browser = new BrowserUI();

        // Симуляція сценарію 1: Відкриття сторінки
        browser.userEntersUrl("google.com");

        // Симуляція сценарію 2: Додавання закладки
        browser.userClicksBookmark();

        // Відкриття іншої сторінки
        browser.userEntersUrl("kpi.ua");

        // Перегляд історії
        browser.userOpensHistory();
    }
}
```

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.

3. 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.