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**»

Основи проектування розгортання

Виконав:

Студент групи ІА-31

Шереметьєв Дмитро

Київ – 2025

Вихідний код: https://github.com/Dimon4ick68/MindMap

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.

Завдання

• Ознай омитись 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.

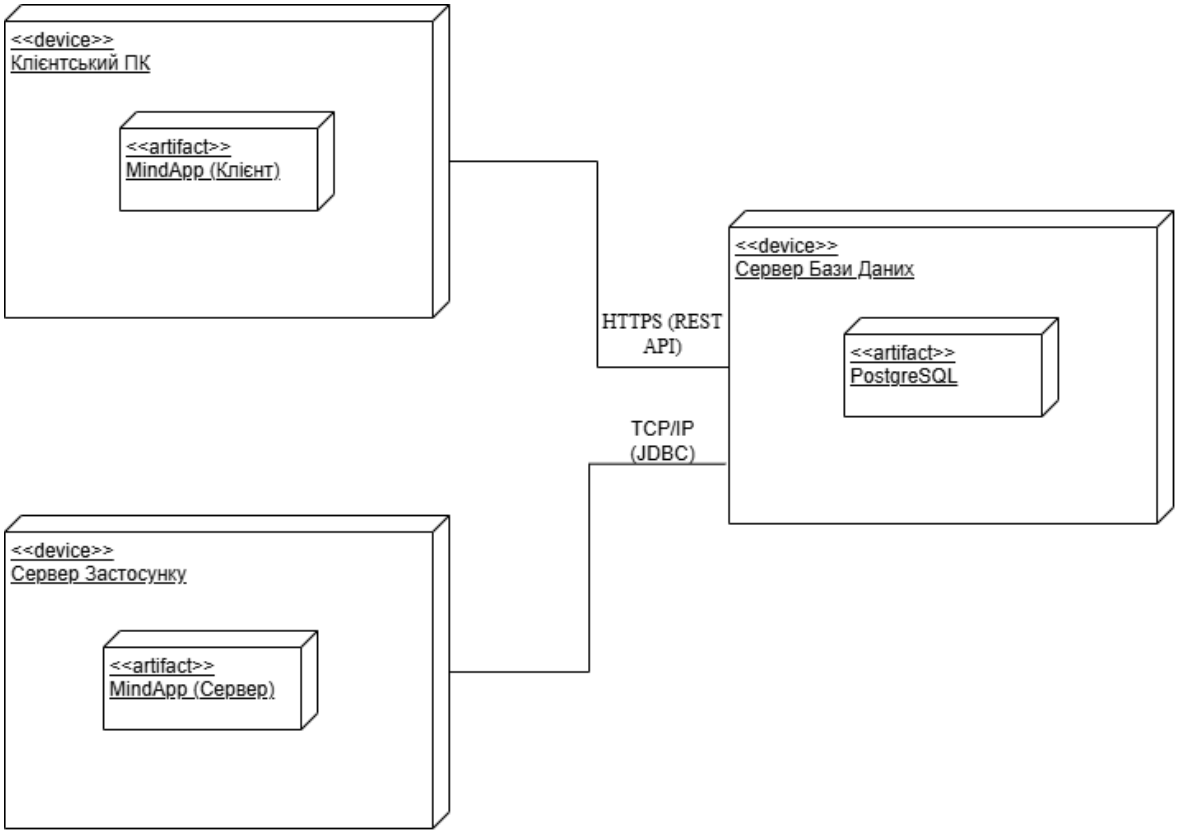
• Розробити діаграму розгортання для проєктованої системи.

• 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 виб орці 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

Опис вузлів та артефактів:

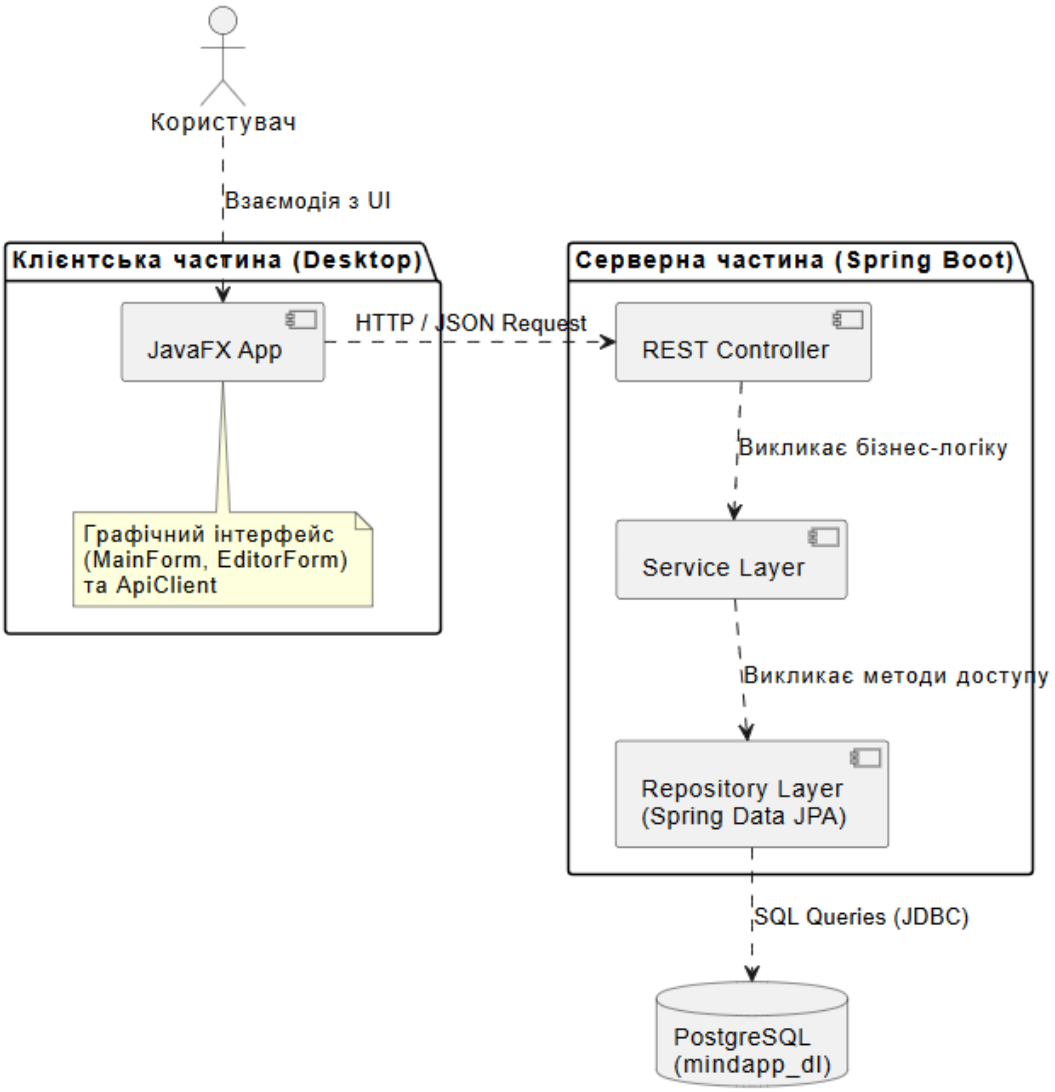
1. **Вузол <<device>> Клієнтський ПК**:
    - **Опис:** Апаратний пристрій кінцевого користувача (персональний комп'ютер або ноутбук).
    - **Артефакт:** <<artifact>> MindApp (Клієнт). Це виконуваний JAR-файл клієнтського додатку, побудованого на технології JavaFX. Він забезпечує графічний інтерфейс користувача та взаємодію з користувачем.
2. **Вузол <<device>> Сервер Застосунку**:

- **Опис:** Виділений сервер або віртуальна машина, де розгорнуто серверну частину системи.
- **Артефакт:** <<artifact>> MindApp (Сервер). Це серверний додаток на базі Spring Boot, що містить бізнес-логіку, REST-контролери та шар доступу до даних. Він запускається у середовищі JVM.

3. **Вузол <<device>> Сервер Бази Даних**:
   - **Опис:** Окремий сервер для зберігання та управління даними.
   - **Артефакт:** <<artifact>> PostgreSQL. Система управління базами даних, яка керує базою даних mindapp_dl, де зберігаються таблиці mind_maps та nodes.

Опис зв'язків:

- **Клієнтський ПК — Сервер Застосунку:** Взаємодія відбувається через протокол **HTTPS** (REST API). Клієнт надсилає HTTP-запити (GET, POST) для отримання та збереження даних у форматі JSON.
- **Сервер Застосунку — Сервер Бази Даних:** Взаємодія реалізована через протокол **TCP/IP** з використанням **JDBC** (Java Database Connectivity). Сервер застосунку виконує SQL-запити до бази даних.



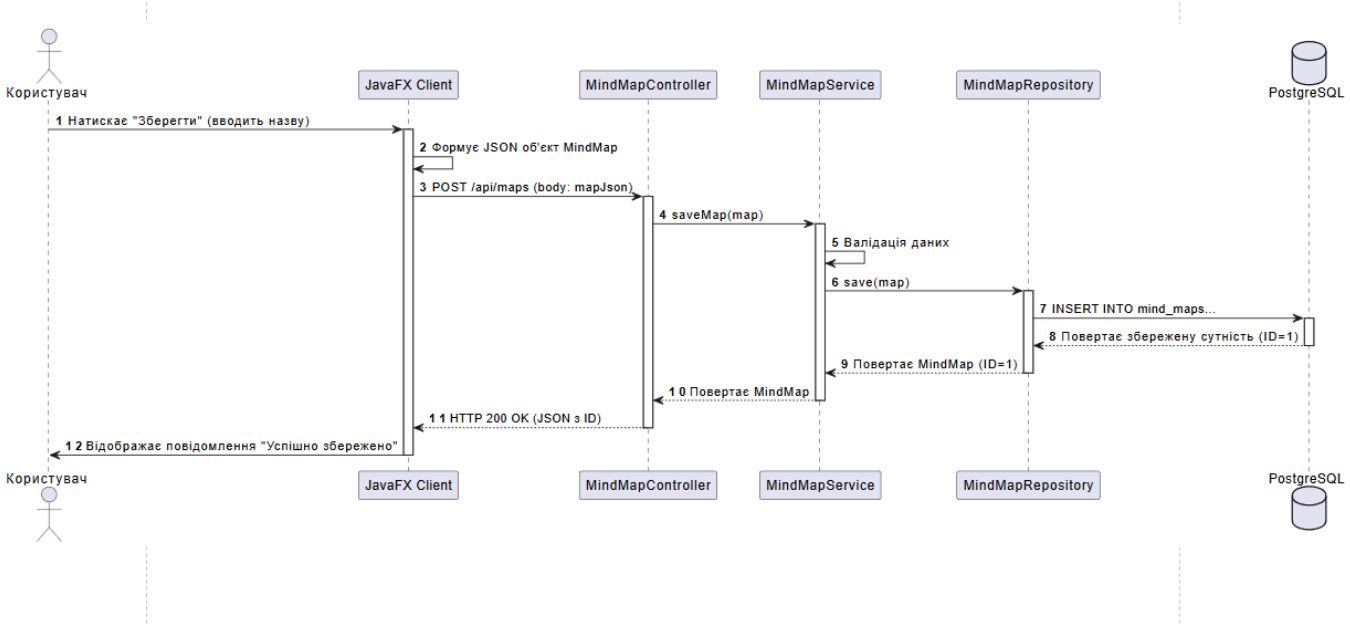
Діаграма компонентів

Опис компонентів:

1. **Клієнтська частина (Desktop):**
    - **JavaFX App**: Головний компонент, що відповідає за візуалізацію (форми, таблиці, canvas). Містить логіку ApiClient для комунікації з сервером.
2. **Серверна частина (Spring Boot):**

- **REST Controller**: Компонент, що приймає вхідні HTTP-запити від клієнта, обробляє їх та повертає відповіді. Слугує точкою входу в бекенд.
- **Service Layer**: Містить основну бізнес-логіку додатку. Виконує валідацію даних та координує роботу між контролером і репозиторієм.
- **Repository Layer (Spring Data JPA)**: Забезпечує абстракцію доступу до даних. Надає інтерфейси для виконання CRUD-операцій над сутностями без необхідності написання чистого SQL.

Опис взаємодії: Користувач взаємодіє з інтерфейсом JavaFX App. Клієнтський додаток надсилає запити до REST Controller. Контролер передає дані у Service Layer для обробки. Сервіс звертається до Repository Layer для збереження або отримання даних з бази даних PostgreSQL.

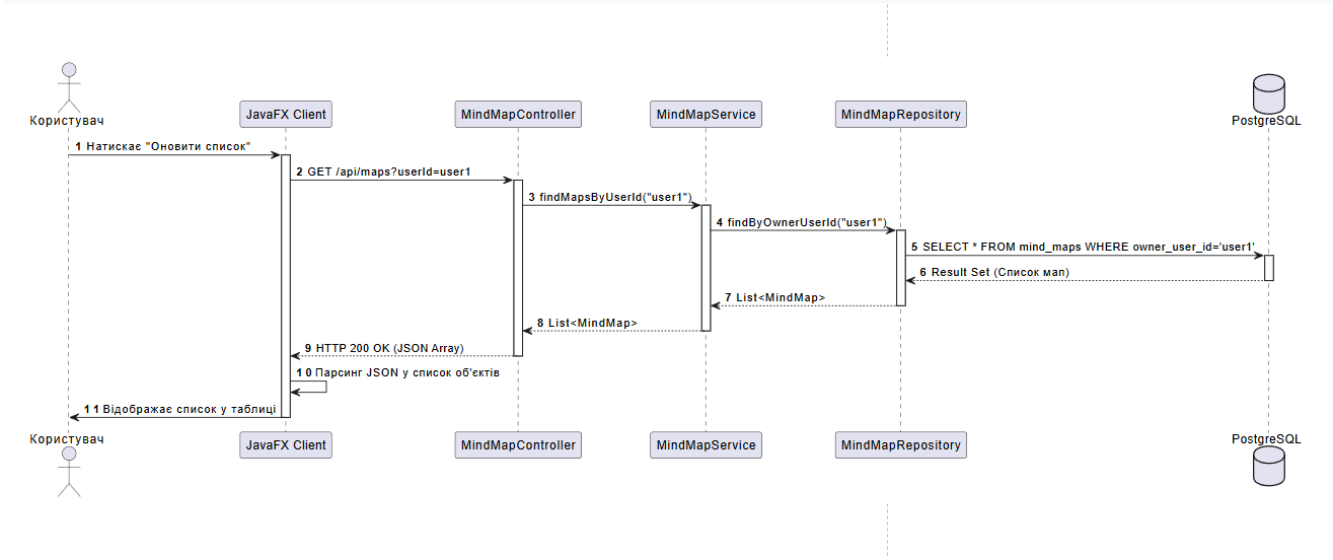


Сценарій 1: "Збереження мапи"

Ця діаграма описує процес створення та збереження нової ментальної карти.

Потік подій:

1. **Користувач** вводить назву мапи на формі редагування та натискає кнопку "Зберегти".
2. **Client (JavaFX)** формує JSON-об'єкт з даними мапи та надсилає HTTP POST запит на сервер (/api/maps).
3. **MindMapController** приймає запит та викликає метод saveMap() у сервісі.
4. **MindMapService** виконує необхідну бізнес-логіку (наприклад, перевірку даних) та викликає метод save() у репозиторії.
5. **MindMapRepository** генерує SQL-запит INSERT і відправляє його до бази даних.
6. **PostgreSQL** зберігає дані та повертає створений запис (з присвоєним ID).
7. Результат повертається по ланцюжку: Репозиторій $\to$ Сервіс $\to$ Контролер.
8. **MindMapController** повертає клієнту відповідь HTTP 200 OK зі збереженим об'єктом.
9. **Client** відображає користувачу повідомлення про успішне збереження.



Сценарій 2: "Отримання списку мап"

Ця діаграма описує процес завантаження списку мап для відображення у головному вікні.

Потік подій:

1. **Користувач** відкриває головне вікно програми або натискає "Оновити список".
2. **Client (JavaFX)** надсилає HTTP GET запит до контролера (/api/maps?userId=...).
3. **MindMapController** звертається до сервісу, викликаючи метод findMapsByUserId().
4. **MindMapService** делегує запит до репозиторію через метод findByOwnerUserId().
5. **MindMapRepository** виконує SQL-запит SELECT до бази даних для вибірки всіх мап вказаного користувача.
6. **PostgreSQL** повертає набір даних (Result Set).
7. **MindMapRepository** перетворює дані у список об'єктів List<MindMap> і повертає його сервісу.
8. **MindMapService** повертає список контролеру.
9. **MindMapController** серіалізує список у JSON і відправляє його клієнту.
10. **Client** десеріалізує JSON та відображає отримані мапи у таблиці інтерфейсу.

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

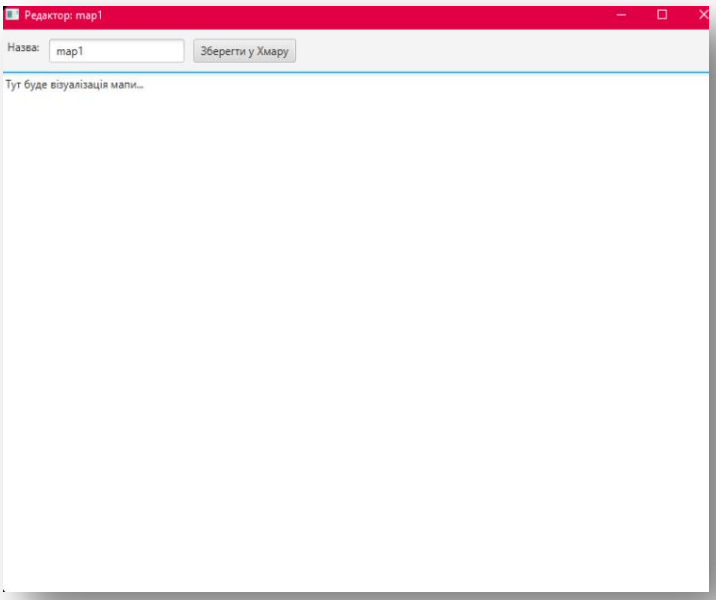
роботи з даними від вводу на формі до збереження їх в БД і подальшій виб орці з БД та відображенням на UI)

Мої дві візуальні форме, а саме:

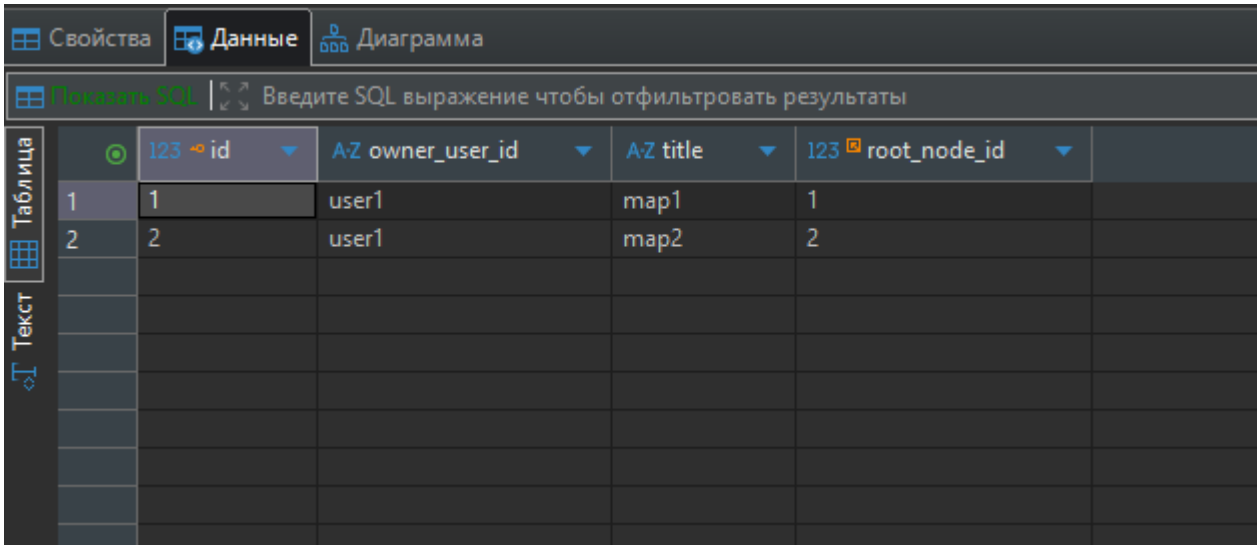
Головна форма

При завантаженні форма звертається до сервера через ApiClient (GET-запит) та отримує список усіх доступних ментальних карт користувача. Отримані дані десеріалізуються з JSON та відображаються у таблиці (TableView).

Форма редактора



Дозволяє користувачу змінити назву карти та ініціювати збереження змін. При натисканні кнопки "Зберегти", дані карти формуються в об'єкт, серіалізуються в JSON і відправляються на сервер (POST-запит).



Після виконання сценарію збереження на клієнті («Створити нову мапу» «Зберегти»), було проведено перевірку стану бази даних PostgreSQL.

За допомогою інструменту адміністрування DBeaver було виконано запит до таблиці mind_maps у базі даних postgres.

Контрольні питання

1. Що собою становить діаграма розгортання?

Діаграма розгортання (Deployment Diagram) — це структурна діаграма UML, яка моделює фізичне розміщення програмних компонентів системи на апаратному забезпеченні. Вона показує, які програмні артефакти (наприклад, .exe або .jar фай ли) встановлюю ться на які фізичні вузли (наприклад, сервери чи комп'ютери) і як ці вузли з'єднані між собою.

 Аналогія: Це як план будівлі, де показано, в якій кімнаті (сервері) стоїть яка техніка (додаток) і як кімнати з'єднані коридорами (мережа).

2. Які бувають види вузлів на діаграмі розгортання?

Вузли (Nodes) — це основні елементи, що представляють обчислювальні ресурси. Існує два основних види:

 Вузол пристрою (Device Node): Представляє фізичне обладнання, таке як сервер, персональний комп'ютер, мобільний телефон аб о будь-який інший апаратний пристрій.

 Вузол середовища виконання (Execution Environment Node): Представляє програмне середовище, яке розміщує та виконує інші програмні компоненти. Приклади: сервер додатків (Tomcat), віртуальна машина Java (JVM), операцій на система або Docker-контей нер.

3. Які бувають зв’язки на діаграмі розгортання?

Основний тип зв'язку — це шлях комунікації (Communication Path). Він зображується у вигляді суцільної лінії між двома вузлами і показує, що вони можуть обмінюватися інформацією. Цей зв'язок часто доповнюють стереотипом, щоб вказати протокол зв'язку, наприклад <<HTTP>>, <<JDBC>> 4. Які елементи присутні на діаграмі компонентів?

 Компонент (Component): Модульна, взаємозамінна частина системи. Це може бути бібліотека (.dll, .jar), виконуваний фай л (.exe) або логічний блок коду (наприклад, "Service Layer").

 Інтерфей с (Interface): Описує набір операцій , які компонент надає іншим (provided interface, "lollipop") або які й ому потрібні від інших (required interface, "socket").

 Порт (Port): Точка взаємодії між компонентом та й ого оточенням.

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

 Залежність (Dependency): Показує, що один компонент залежить від іншого (наприклад, використовує й ого класи або інтерфей си). Зміни в одному компоненті можуть вплинути на інший . Зображується пунктирною стрілкою.

 З'єднувач збірки (Assembly Connector): "З'єднує" необхідний інтерфейс ("socket") одного компонента з наданим інтерфей сом ("lollipop") іншого. Це показує, як компоненти "підключаються" один до одного для спільної роботи.

6. Які бувають види діаграм взаємодії?

Діаграми взаємодії (Interaction Diagrams) показують, як об'єкти співпрацюють з часом. Основні види:

- Діаграма послідовностей (Sequence Diagram)
- Діаграма комунікації (Co mmunication Diagram)
- Діаграма огляду взаємодії (Interaction Overview Diagram)
- Діаграма синхронізації (Timing Diagram)

7. Для чого призначена діаграма послідовностей?

Діаграма послідовностей призначена для візуалізації взаємодії об'єктів у хронологічному порядку. Вона детально показує, яку послідовність повідомлень (викликів методів) об'єкти надсилають один одному для виконання конкретного завдання або сценарію. Головний акцент робиться на часі та порядку викликів.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

- Актор (Actor): Користувач або зовнішня система, що ініціює взаємодію.
- Лінія життя (Lifeline): | Вертикальна пунктирна лінія, що представляє існування об'єкта протягом часу.
- Активація (Activation Bar): Вузький прямокутник на лінії житт я, що показує період, коли об'єкт виконує дію. - Повідомлення (Message): Стрілка між лініями життя, що позначає виклик методу або передачу інформації.
- Фрагмент (Fragment): Рамка, що дозволяє моделювати цикли (loop), умови (alt, opt) та інші складні взаємодії.

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

Вони пов'язані як "що" і "як":

 Діаграма варіантів використання показує, ЩО система робить для користувача (наприклад, "Створити документ").

 Діаграма послідовностей показує, ЯК система це робить всередині, деталізуючи один конкретний сценарій цього варіанту використання (наприклад, покроковий процес створення документа).

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

Вони пов'язані як "структура" і "поведінка":

 Діаграма класів показує статичну структуру системи — які класи існують та які методи вони мають (креслення акторів).

 Діаграма послідовностей показує динамічну поведінку — як об'єкти (екземпляри) цих класів взаємодіють між собою, викликаючи методи один одного в реальному часі (сцена, де актори грають свої ролі).