Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут» імені Ігоря Сікорського
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота № 3

із дисципліни: «Технології розроблення програмного забезпечення»

на тему: «ДІАГРАМА РОЗГОРТАННЯ. ДІАГРАМА КОМПОНЕНТІВ. ДІАГРАМА ВЗАЄМОДІЙ ТА ПОСЛІДОВНОСТЕЙ**»**

Виконав:
студент групи ІА-34
Чунаков А.Р.
Перевірив:
Мягкий Михайло Юрійович

Київ-2025

Мета: Навчитися будувати діаграму розгортання, діаграму компонентів, діаграму взаємодій та послідовностей

Короткі теоретичні відомості:

Діаграма розгортання (Deployment Diagram) — це структурна діаграма UML, яка візуалізує фізичну архітектуру програмної системи, показуючи розподіл програмних артефактів (файлів, бібліотек) по апаратних вузлах (серверах, робочих станціях) та зв'язки між ними. Вона використовується для того, щоб зрозуміти, на якому обладнанні працюватиме програма, як різні частини системи з'єднані мережею та які протоколи комунікації використовуються, що є критично важливим для системних адміністраторів та інженерів з розгортання.

Діаграма компонентів (Component Diagram) — це структурна діаграма UML, що відображає модульну структуру програмного забезпечення, розбиваючи систему на логічні або фізичні компоненти та показуючи залежності між ними через інтерфейси. Її головна мета — дати високорівневе уявлення про архітектуру додатка, дозволяючи розробникам побачити, з яких великих блоків (модулів, бібліотек, шарів) складається система і як зміни в одному компоненті можуть вплинути на інші, що допомагає керувати складністю проєкту.

Діаграма послідовностей (Sequence Diagram) — це поведінкова діаграма взаємодії UML, яка моделює динамічну поведінку системи, показуючи часову послідовність обміну повідомленнями між об'єктами для виконання конкретного сценарію використання. Вона необхідна для детального проєктування логіки методів та функцій, дозволяючи виявити помилки в алгоритмах взаємодії та переконатися, що всі необхідні кроки бізнес-процесу виконуються у правильному порядку.

1. Розробка діаграми розгортання для проектованої системи:

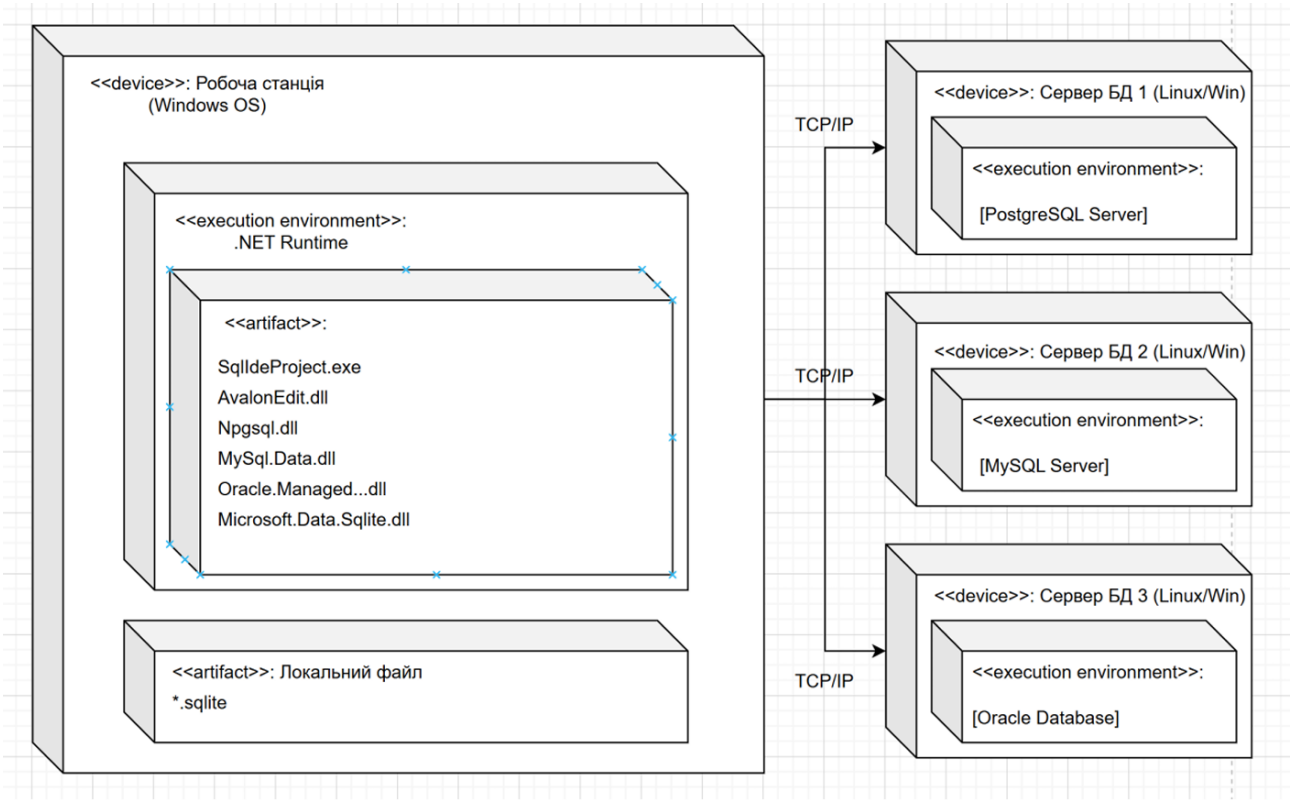

Рис. 1.1 – Діаграма розгортання системи

Ця діаграма розгортання візуалізує фізичну архітектуру системи SQL IDE, показуючи, як компоненти програмного забезпечення (артефакти) розміщуються на фізичних пристроях та як вони взаємодіють.

1) Вузол "Робоча станція" (Клієнт)

Ліворуч зображено основний клієнтський вузол — <<device>>: Робоча станція (Windows OS). Це персональний комп'ютер користувача, на якому безпосередньо запускається додаток. На цьому пристрої розміщено два ключові компоненти:

1) <<execution environment>>: .NET Runtime: Це середовище виконання, необхідне для запуску будь-якого C#/.NET додатку. Воно "хостить" (містить у собі) усі основні артефакти програми:
   - <<artifact>>: SqlIdeProject.exe: Головний виконуваний файл нашого SQL IDE.
   - Набір .dll бібліотек: Це залежності, необхідні для роботи програми. Вони включають AvalonEdit.dll (для редактора коду) та повний набір драйверів баз даних (Npgsql.dll, MySql.Data.dll, Oracle.Managed...dll, Microsoft.Data.Sqlite.dll), що є ключовим для реалізації патерну Bridge.

2) <<artifact>>: Локальний файл *.sqlite: Цей артефакт зображений окремо від середовища .NET Runtime, оскільки він представляє собою файл даних на жорсткому диску (наприклад, main.sqlite). Наявність Microsoft.Data.Sqlite.dll у середовищі виконання передбачає, що програма буде взаємодіяти з цим файлом через файлову систему (File I/O).

2) Вузли "Сервери БД" (Сервери)

Праворуч зображено три незалежні серверні вузли. Вони представляють сервери баз даних, до яких підключається наш додаток. Вони можуть бути як фізичними серверами в мережі, так і віртуальними машинами, або навіть запущеними локально (localhost).

- <<device>>: Сервер БД 1 (Linux/Win): Містить середовище виконання [PostgreSQL Server].
- <<device>>: Сервер БД 2 (Linux/Win): Містить середовище виконання [MySQL Server].
- <<device>>: Сервер БД 3 (Linux/Win): Містить середовище виконання [Oracle Database].

3) Зв'язки (Комунікації)

Діаграма показує три чіткі зв'язки типу TCP/IP.

Ці зв'язки виходять із середовища .NET Runtime на клієнті та вказують на кожне із трьох серверних середовищ (PostgreSQL, MySQL, Oracle). Це демонструє, що наш додаток спілкується з цими базами даних через стандартний мережевий протокол (на відміну від SQLite, де взаємодія відбувається через файлову систему).

2. Розробка діаграму компонентів для проектованої системи.

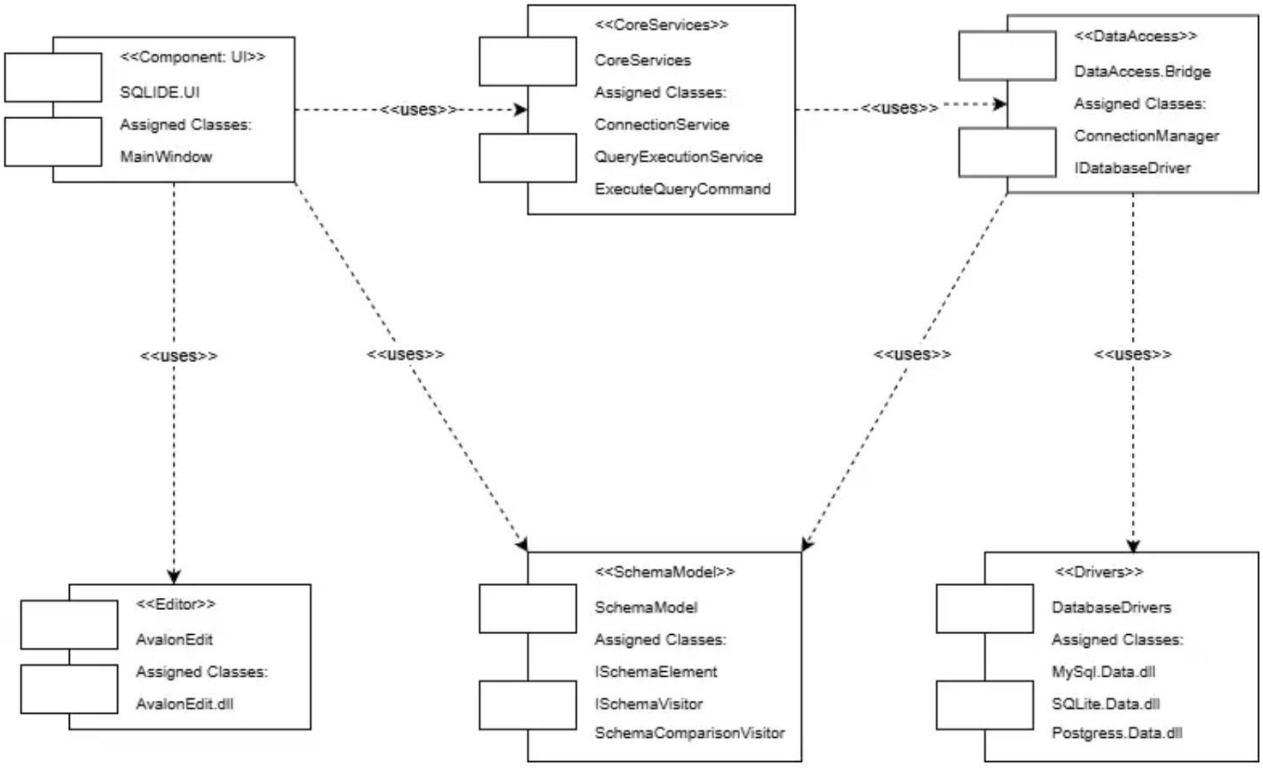

Рис. 1.2 – Діаграма компонентів системи

Ця діаграма компонентів ілюструє архітектуру системи SQL IDE, розділяючи її на логічні, функціональні блоки (компоненти) та показуючи залежності між ними.

Компоненти Системи

Діаграма представляє шість основних компонентів, які можна розділити на внутрішні (розроблені в рамках проєкту) та зовнішні (сторонні бібліотеки).

Внутрішні Компоненти:

- <<Component: UI>> SQLIDE.UI: Це компонент інтерфейсу користувача, "обличчя" програми. Йому призначено клас MainWindow, який відповідає за відображення вікна та обробку дій користувача.
- <<CoreServices>> CoreServices: Це "мозок" системи, що містить основну бізнес-логіку. Йому призначені класи, що реалізують ключові патерни: ConnectionService (Singleton), QueryExecutionService (Observer) та ExecuteQueryCommand (Command).
- <<DataAccess>> DataAccess.Bridge: Компонент, що реалізує патерн "Міст" (Bridge). Він служить абстрактним шаром для доступу до даних. Йому призначені класи ConnectionManager та інтерфейс IDatabaseDriver, які ізолюють решту системи від конкретних СУБД.
- <<SchemaModel>> SchemaModel: Компонент, що реалізує патерн "Відвідувач" (Visitor). Він містить класи для опису структури бази даних

(ISchemaElement), інтерфейс "відвідувача" (ISchemaVisitor) та його конкретні реалізації (наприклад, SchemaComparisonVisitor).

Зовнішні Компоненти (Бібліотеки):

- <<Editor>> AvalonEdit: Це зовнішній компонент (бібліотека AvalonEdit.dll), який надає функціонал повноцінного текстового редактора з підсвічуванням синтаксису SQL.
- <<Drivers>> DatabaseDrivers: Це набір зовнішніх бібліотек .dll (наприклад, MySql.Data.dll, Postgress.Data.dll), які є конкретними реалізаціями інтерфейсу IDatabaseDriver.

2. Залежності (<<uses>>)

Пунктирні стрілки (<<uses>>) показують залежності між компонентами:

- UI - CoreServices: Інтерфейс користувача викликає сервіси для виконання дій (наприклад, ExecuteQueryButton_Click викликає QueryExecutionService.Submit()).
- UI --> AvalonEdit: Інтерфейс користувача безпосередньо використовує (відображає) компонент AvalonEdit як текстове поле.
- UI --> SchemaModel: UI використовує "відвідувачів" з цього компонента (наприклад, SchemaTextRendererVisitor) для відображення результатів аналізу схеми.
- CoreServices - DataAccess.Bridge: Сервіси (зокрема, ExecuteQueryCommand) використовують ConnectionManager для виконання запитів, не знаючи, яка СУБД знаходиться "під капотом".
- DataAccess.Bridge - Drivers: Компонент "Міст" (через інтерфейс IDatabaseDriver) покладається на конкретні реалізації з бібліотек драйверів для фактичного зв'язку з БД.
- DataAccess.Bridge - SchemaModel: Компонент DataAccess (через свої драйвери) створює об'єкти моделі схеми (наприклад, TableElement), коли викликається метод GetSchema().

3. Розробка діаграми послідовностей для проектованої системи.

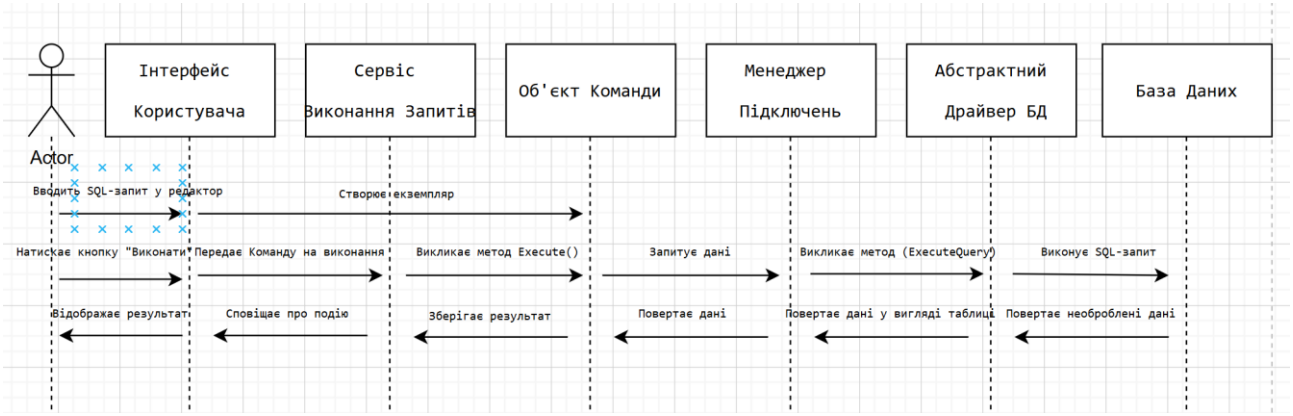

Рис. 1.3 –Діаграма послідовностей для сценарію Виконання SQL-запиту

Детальний опис кроків:

1. Користувач вводить текст SQL-запиту у текстовий редактор в інтерфейсі.
2. Користувач натискає кнопку "Виконати запит".
3. Інтерфейс Користувача (UI) створює спеціальний "Об'єкт Команди" (патерн Command), інкапсулюючи в ньому текст запиту.
4. UI передає цей об'єкт централізованому "Сервісу Виконання Запитів".
5. Сервіс викликає метод виконання у переданої команди.
6. Команда, у свою чергу, звертається до активного "Менеджера Підключень" і просить його виконати запит.
7. Менеджер Підключень не знає про специфіку бази даних; він делегує виконання "Абстрактному Драйверу БД" (патерн Bridge).
8. Конкретний драйвер (наприклад, драйвер для PostgreSQL або SQLite) формує і відправляє запит до фізичної Бази Даних.
9. База Даних повертає результат.
10. Драйвер форматує результат у стандартизований вигляд (наприклад, таблицю даних) і повертає його Менеджеру Підключень, а той – Об'єкту Команди.
11. "Сервіс Виконання Запитів" отримує контроль назад і генерує подію (патерн Observer), сповіщаючи всіх підписаних слухачів про те, що запит завершено (успішно або з помилкою), і додаючи до події отримані дані.
12. Інтерфейс Користувача, будучи підписаним на цю подію, отримує сповіщення.

13. UI оновлює відповідні елементи – заповнює таблицю результатів та оновлює текст у рядку стану, показуючи результат користувачу.

Висновок:

Розробка комплекту діаграм (розгортання, компонентів та послідовностей) стала етапом проєктування, який дозволив всебічно описати архітектуру системи SQL IDE від фізичного розміщення до логіки взаємодії об'єктів. Діаграма розгортання візуалізувала фізичну клієнт-серверну архітектуру та протоколи зв'язку, підтвердивши відповідність нефункціональним вимогам; діаграма компонентів забезпечила чіткий поділ системи на незалежні модулі (UI, Сервіси, Шар даних), що є основою гнучкості та розширюваності; а діаграма послідовностей дозволила верифікувати коректність динамічної поведінки системи та правильність реалізації патернів Command і Bridge, гарантуючи, що всі компоненти взаємодіють узгоджено та без помилок логіки.

Контрольні питання:

1. Що собою становить діаграма розгортання (Deployment Diagram)? Діаграма розгортання — це структурна діаграма UML, яка візуалізує фізичну архітектуру системи, показуючи, як програмні компоненти (артефакти) розподілені між апаратними засобами (вузлами). Вона відображає топологію системи, зв'язки між обчислювальними пристроями та середовищами виконання, що дозволяє зрозуміти, де саме працюватиме код і як компоненти комунікують на фізичному рівні.

2. Які бувають види вузлів на діаграмі розгортання? Вузли на діаграмі розгортання поділяються на два основні види: Пристрої (Devices) та Середовища виконання (Execution Environments). Пристрої — це фізичне обладнання з обчислювальною потужністю (сервери, комп'ютери, мобільні телефони, маршрутизатори), а середовища виконання — це програмне забезпечення, що працює на цих пристроях і "хостить" інші компоненти (операційні системи, .NET CLR, Java Virtual Machine, веб-сервери, сервери баз даних).

3. Які бувають зв’язки на діаграмі розгортання? Основним типом зв'язку на діаграмі розгортання є Шлях комунікації (Communication Path), який зображується суцільною лінією і показує, що вузли можуть обмінюватися повідомленнями. Ці зв'язки зазвичай стереотипуються для зазначення конкретного протоколу або технології взаємодії, наприклад, <<TCP/IP>>,

<<HTTP>>, <<JDBC>> або <<RMI>>, що дає чітке уявлення про спосіб передачі даних між вузлами.

4. Які елементи присутні на діаграмі компонентів? На діаграмі компонентів основними елементами є Компоненти (модульні частини системи, що інкапсулюють свій вміст), Інтерфейси (надані та необхідні, що визначають точки взаємодії компонента з оточенням), Порти (специфічні точки доступу до внутрішньої структури компонента) та Зв'язки (залежності та реалізації). Також можуть використовуватися Артефакти для відображення фізичних файлів, що реалізують компоненти.

5. Що становлять собою зв’язки на діаграмі компонентів? Зв'язки на діаграмі компонентів відображають логічні відносини між модулями. Найпоширенішими є Залежність (Dependency) (пунктирна стрілка), яка показує, що один компонент використовує або потребує іншого, та Реалізація (Realization), яка показує, що компонент реалізує певний інтерфейс. Також використовуються зв'язки типу "Assembly Connector" (з'єднання "куля-гніздо" між наданим і необхідним інтерфейсами) для моделювання складання системи з частин.

6. Які бувають види діаграм взаємодії (Interaction Diagrams)? В UML існує чотири основні види діаграм взаємодії, кожна з яких фокусується на різних аспектах комунікації: Діаграма послідовностей (Sequence Diagram) (акцент на часовій послідовності повідомлень), Діаграма комунікації (Communication Diagram) (акцент на структурних зв'язках між об'єктами), Діаграма огляду взаємодії (Interaction Overview Diagram) (поєднує діаграми активності та послідовності для показу потоку керування) та Діаграма синхронізації (Timing Diagram) (акцент на зміні стану об'єктів у часі).

7. Для чого призначена діаграма послідовностей (Sequence Diagram)? Діаграма послідовностей призначена для моделювання динамічної поведінки системи в рамках одного конкретного сценарію використання. Вона детально візуалізує часовий порядок обміну повідомленнями між об'єктами, дозволяючи розробникам побачити, які методи викликаються, в якій послідовності, які дані передаються і як довго живуть об'єкти, що є незамінним інструментом для проєктування логіки складних процесів.

8. Які ключові елементи можуть бути на діаграмі послідовностей? Ключовими елементами є Лінії життя (Lifelines) (вертикальні пунктирні лінії, що

представляють існування об'єкта в часі), Актори (зовнішні ініціатори), Фокус керування (Activation Bars) (прямокутники на лініях життя, що показують період активності об'єкта), та Повідомлення (Messages) (стрілки між лініями життя: синхронні, асинхронні, відповіді, створення/знищення об'єктів). Також використовуються Комбіновані фрагменти (Combined Fragments) для відображення циклів (loop), умов (alt, opt) та паралельного виконання (par).

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання? Діаграма послідовностей є деталізацією одного конкретного сценарію з діаграми варіантів використання. Якщо діаграма варіантів використання показує що система робить (наприклад, "Виконати замовлення"), то діаграма послідовності показує як саме це відбувається крок за кроком, розкриваючи внутрішню логіку взаємодії об'єктів, необхідну для реалізації цього варіанту використання.

10. Як діаграми послідовностей пов’язані з діаграмами класів? Діаграми послідовностей і класів тісно пов'язані: діаграма класів описує статичну структуру (типи об'єктів та їх методи), а діаграма послідовностей показує динамічні виклики цих методів. Кожне повідомлення на діаграмі послідовностей повинно відповідати реальному методу (операції) у класі отримувача на діаграмі класів, а об'єкти (Lifelines) є екземплярами цих класів, що забезпечує узгодженість між статичною та динамічною моделями системи.