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.
Перевірил:
Мягкий Михайло Юрійович

Київ-2025

Завдання.

1. Ознайомитися з короткими теоретичними відомостями.

2. Розробити діаграму розгортання для проектованої системи.

3. Розробити діаграму компонентів для проектованої системи.

4. Розробити діаграму послідовностей для проектованої системи.

5. Скласти звіт про виконану роботу.

1. Розробка діаграми розгортання для проектованої системи:

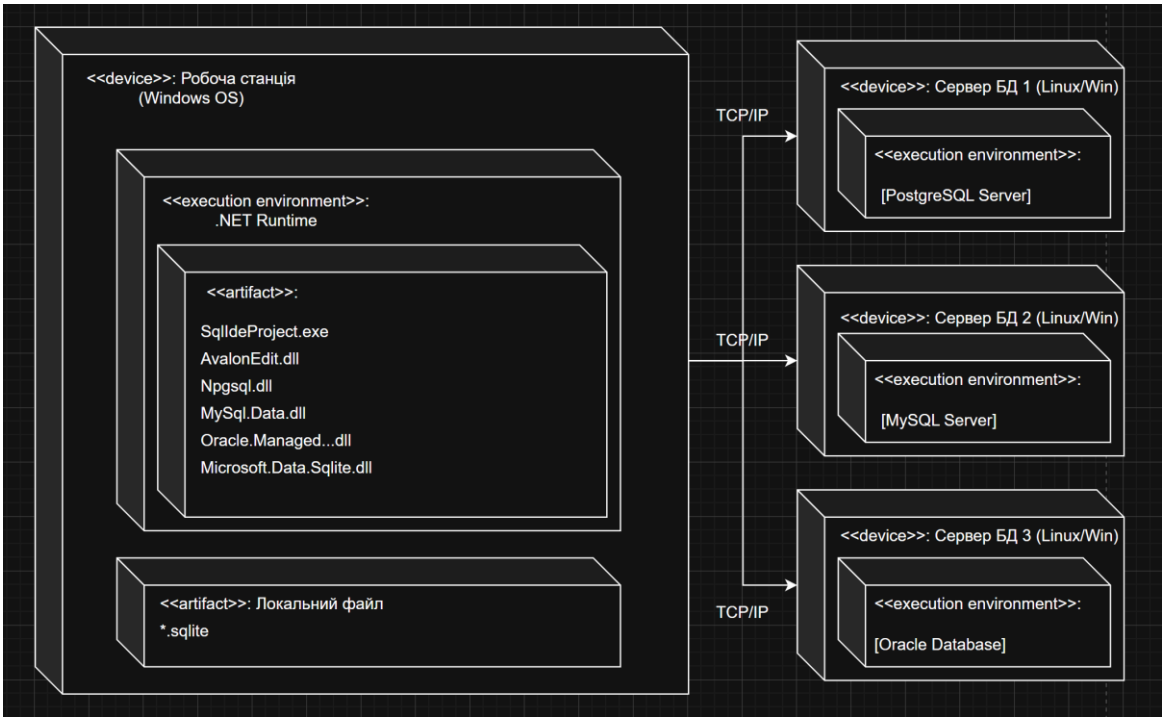

Рис. 1.1 – Діаграма розгортання системи

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.

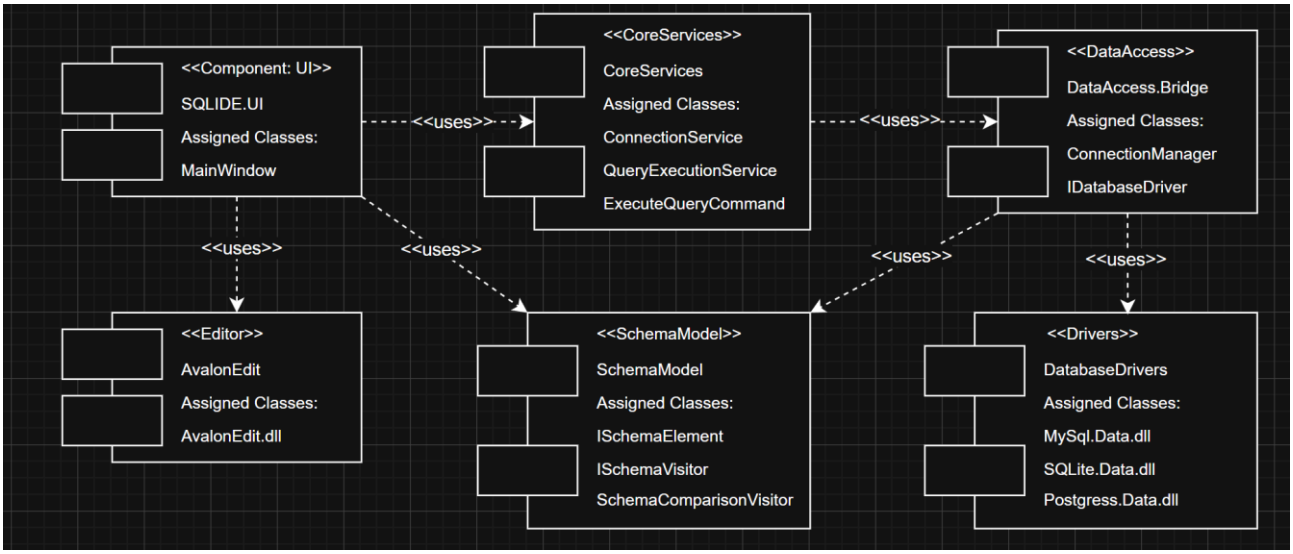

Рис. 1.2 – Діаграма компонентів системи

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.

Сценарій: Виконання SQL-запиту

Назва: Виконання запиту

Актор: Користувач БД

Мета: Виконати SQL-запит до підключеної БД та отримати результат.

Опис: Це найскладніший сценарій, оскільки він задіює всю основну архітектуру системи та декілька архітектурних патернів (Команда, Спостерігач, Міст).
Через свою складність він представлений у вигляді діаграми послідовності.

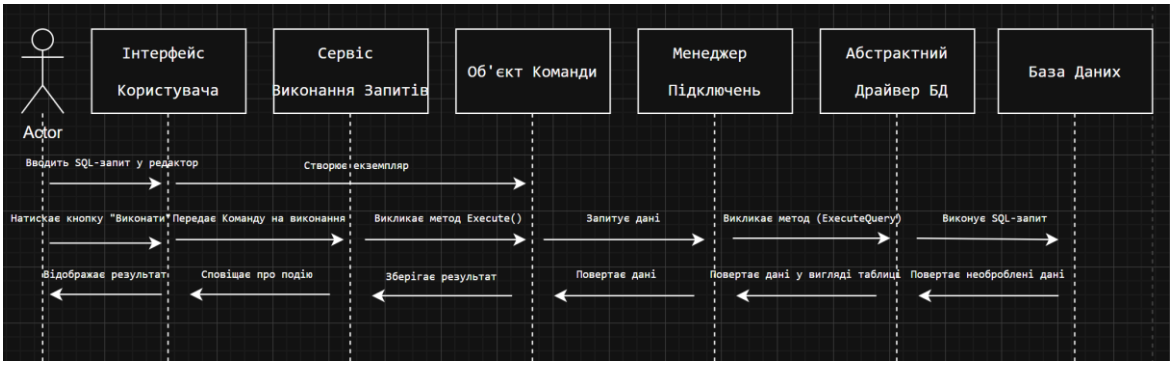

Рис. 1.3 –Діаграма послідовностей для сценарію Виконання SQL-запиту

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: У ході лабораторної роботи я реалізував частину програми описану у попередній роботі, а також побудував діаграми розгортання, компонентів та послідовностей системи.