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 № 9

із дисципліни: «Технології розроблення програмного забезпечення»

на тему: Взаємодія компонентів системи

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

Київ-2025

Мета: Вивчити структуру шаблонів «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR» та навчитися застосовувати їх в реалізації програмної системи.

Короткі теоретичні відомості:

Клієнт-серверна архітектура — це модель побудови розподілених систем, де функціональність розділена між двома типами вузлів: клієнтами, які ініціюють запити на отримання даних або послуг, та серверами, які ці запити обробляють, зберігають дані та надсилають відповіді. Ця модель забезпечує централізоване керування ресурсами та безпекою на стороні сервера, дозволяючи при цьому мати безліч клієнтів з різними інтерфейсами (веб, мобільні, десктопні), що спрощує масштабування та адміністрування системи.

Архітектура Peer-to-Peer (однорангова) архітектура — це децентралізована мережева модель, де всі учасники (вузли або "піри") мають рівні права і можуть виступати одночасно як клієнтами, так і серверами, обмінюючись ресурсами без посередництва центрального сервера. Така структура забезпечує високу відмовостійкість та масштабованість, оскільки вихід з ладу одного вузла не зупиняє роботу всієї мережі, а додавання нових учасників лише збільшує загальну потужність та доступність ресурсів системи.

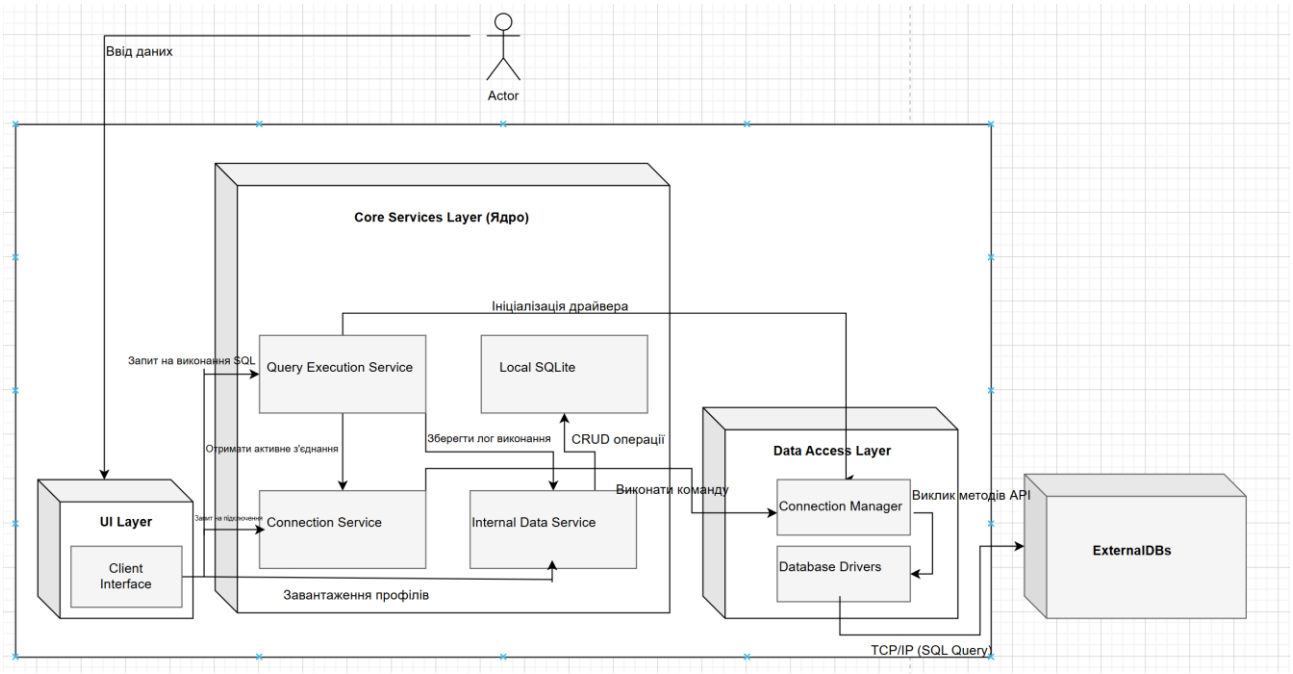
Сервіс-орієнтована архітектура (SOA) — це підхід до проєктування програмного забезпечення, де система будується як набір слабко пов'язаних, автономних сервісів, які взаємодіють між собою через чітко визначені мережеві протоколи та інтерфейси (часто через корпоративну шину сервісів — ESB). Головною метою SOA є забезпечення повторного використання бізнес-логіки та інтеграція різнорідних систем, що дозволяє великим підприємствам гнучко компонувати складні бізнес-процеси з існуючих сервісів незалежно від платформи їх реалізації.

Мікросервісна архітектура — це сучасна еволюція SOA, де додаток розробляється як сукупність невеликих, незалежних сервісів, кожен з яких відповідає за одну конкретну бізнес-функцію, має власну базу даних та розгортається окремо. Такий підхід дозволяє командам розробників працювати автономно, використовувати різні технологічні стеки для різних задач, а також масштабувати лише ті частини системи, які цього потребують, що значно підвищує гнучкість розробки та стійкість додатка до збоїв.

Обрана тема лабораторної роботи: **SQL IDE**

9. **Sql IDE** (singleton, command, observer, bridge, visitor, SOA)

Інтегроване середовище розробки скриптів SQL повинна вміти працювати з більшістю сучасних баз даних (Microsoft SQL server, Oracle, IBM DB2, MySQL, PostgreSQL), виділяти синтаксичні одиниці конструкцій, виконувати і показувати результат їх виконання, отримувати схеми даних з джерела (список таблиць, стовпців, ключів і т.д.), порівнювати схеми даних з різних джерел.



Діаграма яка представляє спроєктовану архітектуру

Хід роботи:

Фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури:

Фрагмент 1: Реалізація патерну Bridge (Підтримка різних СУБД)

Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/DataAccess/ConnectionManager.cs

Цей фрагмент демонструє, як система відокремлює абстракцію від реалізації. Це дозволяє системі працювати з будь-якою базою даних через єдиний інтерфейс.

Цей код є ядром гнучкості системи. Він доводить, що основна логіка програми не залежить від конкретних бібліотек (Npgsql, MySql.Data), а працює через абстрактний контракт IDatabaseDriver.

Фрагмент 2: Реалізація патерну Singleton (Керування станом)

Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/Services/ConnectionService.cs

Цей фрагмент показує механізм централізованого керування підключеннями.

Код демонструє класичну реалізацію Thread-Safe Singleton. Це гарантує, що всі частини програми використовують один і той самий пул з'єднань, що критично для цілісності даних та керування ресурсами.

Фрагмент 3: Реалізація патерну Command (Інкапсуляція запиту)

Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/Commands/ExecuteQueryCommand.cs

Фрагмент демонструє, як запит користувача перетворюється на об'єкт.

Цей код показує розрив зв'язку між ініціатором (кнопкою) та виконавцем (БД). Завдяки цьому підходу система може легко зберігати історію запитів, оскільки кожен запит є самостійним об'єктом із даними.

Фрагмент 4: Реалізація патерну Repository (Доступ до внутрішніх даних)

Посилання на код - https://github.com/1chuny/SQL-IDE/blob/main/DataAccess/Repositories/Sqlite/SqliteConnectionProfileRepository.cs

Цей код показує, як бізнес-логіка ізолюється від деталей збереження даних в SQLite

Фрагмент демонструє "чисту" реалізацію доступу до даних. Основна програма викликає метод Add(), не знаючи, що всередині відбувається підключення до файлу SQLite і виконання SQL-інструкцій. Це дозволяє змінювати спосіб зберігання налаштувань без зміни логіки програми.

Висновок

У ході роботи було детально розглянуто та проаналізовано основні архітектурні стилі розподілених систем, що дозволило обґрунтовано обрати клієнт-серверну архітектуру для реалізації проєкту SQL IDE. Вивчення принципів SOA та мікросервісної архітектури поглибило розуміння важливості модульності, слабкої зв’язності компонентів та використання стандартизованих протоколів взаємодії, що було частково втілено в проєкті через виділення сервісного шару (Service Layer) та використання патерну Bridge для абстрагування доступу до даних. Отримані знання дозволяють проєктувати масштабовані та гнучкі системи, які легко адаптуються до змін вимог та інтегруються з різними зовнішніми сервісами.

Контрольні питання

1. Що таке клієнт-серверна архітектура? Клієнт-серверна архітектура — це модель побудови розподілених систем, де функціональне навантаження розділене між постачальниками послуг (серверами) і замовниками послуг (клієнтами). Клієнти ініціюють запити на отримання даних або виконання операцій, а сервери обробляють ці запити, виконують обчислення, звертаються до баз даних і надсилають результат назад, що дозволяє централізувати зберігання даних і логіку обробки, забезпечуючи при цьому доступ до системи з багатьох робочих місць.

2. Розкажіть про сервіс-орієнтовану архітектуру (SOA). Сервіс-орієнтована архітектура (SOA) — це підхід до розробки програмного забезпечення, в якому додатки будуються з набору дискретних, слабко пов'язаних сервісів, що взаємодіють між собою через чітко визначені інтерфейси та протоколи, часто через мережу. Основна ідея SOA полягає в повторному використанні бізнес-функцій, інкапсульованих у сервісах, що дозволяє швидко створювати нові додатки шляхом комбінування існуючих компонентів, незалежно від платформи чи мови програмування, на якій вони реалізовані.

3. Якими принципами керується SOA? SOA керується набором принципів, ключовими з яких є: слабка зв'язність (сервіси мають мінімальні залежності один від одного), абстракція (внутрішня логіка прихована, доступний лише інтерфейс), повторне використання (сервіси проєктуються для використання у багатьох бізнес-процесах), автономність (сервіси контролюють власне середовище виконання) та сумісність (здатність взаємодіяти незалежно від технологічної платформи через стандартні протоколи).

4. Як між собою взаємодіють сервіси в SOA? У SOA сервіси взаємодіють за допомогою повідомлень, що передаються через стандартизовані протоколи (найчастіше SOAP або REST over HTTP), і часто використовують проміжний шар — корпоративну сервісну шину (Enterprise Service Bus, ESB). ESB виступає як комунікаційний хаб, який маршрутизує повідомлення, трансформує формати даних (наприклад, з XML у JSON) і забезпечує оркестрацію викликів, дозволяючи сервісам спілкуватися, навіть якщо вони використовують різні транспортні протоколи.

5. Як розробники взнають про існуючі сервіси і як робити до них запити? Для виявлення сервісів у SOA використовується реєстр сервісів (Service Registry) або каталог, де публікуються описи доступних сервісів, їхні інтерфейси (контракти, наприклад, WSDL для SOAP або Swagger/OpenAPI для REST) та політики використання. Розробники знаходять потрібний сервіс у реєстрі, вивчають його контракт, який описує доступні методи та формат даних, і на основі цього формують клієнтський код для виконання запитів.

6. У чому полягають переваги та недоліки клієнт-серверної моделі? Перевагами є централізоване керування даними та безпекою, легкість обслуговування

(оновлення сервера автоматично доступне всім клієнтам) і масштабованість серверної частини. Недоліками є наявність єдиної точки відмови (якщо сервер падає, робота зупиняється), можливі перевантаження мережі при великій кількості запитів та висока вартість серверного обладнання і його адміністрування.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії (P2P)? Перевагами P2P є висока відмовостійкість (немає центрального сервера, вихід з ладу одного вузла не критичний), легка масштабованість (кожен новий клієнт додає ресурси в мережу) та відсутність потреби у дорогому серверному обладнанні. Недоліками є складність адміністрування та забезпечення безпеки, неможливість гарантувати доступність конкретного файлу чи ресурсу в будь-який момент часу та ускладнена логіка пошуку даних у мережі.

8. Що таке мікросервісна архітектура? Мікросервісна архітектура — це стиль проєктування, де єдиний додаток розробляється як набір невеликих сервісів, кожен з яких працює у власному процесі, відповідає за окрему бізнес-функцію (Bounded Context) і може бути розгорнутий незалежно. Це дозволяє використовувати різні технології для різних сервісів, спрощує масштабування (можна масштабувати лише "гарячі" сервіси) та прискорює розробку, оскільки невеликі команди можуть працювати паралельно над різними мікросервісами.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі? Для синхронної взаємодії найчастіше використовують легковагові протоколи HTTP/HTTPS (стиль REST) або gRPC (для високопродуктивного міжсервісного зв'язку). Для асинхронної взаємодії широко застосовуються протоколи черг повідомлень, такі як AMQP (RabbitMQ) або Kafka, що дозволяє зменшити зв'язність сервісів і підвищити стійкість системи до пікових навантажень.

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, коли ми в проєкті між шаром веб-контролерів та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів? Строго кажучи, ні, це називається Service Layer Pattern (патерн Шару Сервісів) у рамках монолітної або шаруватої архітектури, а не повноцінною SOA. Хоча це використовує принципи розділення відповідальності, характерні для SOA, справжня Сервіс-Орієнтована Архітектура передбачає, що сервіси є розподіленими, автономними компонентами, які можуть бути розгорнуті на різних серверах і використовуватися різними, незалежними додатками, а не просто класами всередині однієї програми.