Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота № 9

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Виконав
Студент групи ІА-31:
Губар Б. О.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

1. **Мета:**

Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

2. **Теоретичні відомості**

Клієнт-серверна архітектура — модель, де клієнт відповідає за взаємодію з користувачем, а сервер — за зберігання та обробку даних. Тонкий клієнт (наприклад, вебзастосунки) передає більшість операцій на сервер, спрощуючи оновлення. Товстий клієнт (мобільні або десктопні програми) виконує логіку локально, зменшуючи навантаження на сервер і дозволяючи працювати офлайн. SPA (Single Page Application) — проміжний варіант: логіка на клієнті, але робота можлива лише з підключенням до сервера. Типова структура включає три рівні: клієнтський (інтерфейс), спільний (middleware) і серверний (бізнес-логіка та дані).

Peer-to-Peer (P2P) — децентралізована модель, де кожен вузол одночасно є клієнтом і сервером. Усі учасники рівноправні, обмінюються ресурсами без центрального сервера (наприклад, BitTorrent, блокчейн, Skype). Недоліки: складність забезпечення безпеки, синхронізації та пошуку даних у великих мережах.

Сервіс-орієнтована архітектура (SOA) — модульний підхід, де система складається з незалежних сервісів зі стандартизованими інтерфейсами (HTTP, SOAP, REST). Сервіси виконують конкретні бізнес-функції, обмінюються повідомленнями і можуть бути інтегровані через Enterprise Service Bus (ESB). SOA стала основою для мікросервісів.

Мікросервісна архітектура — створення додатків як набору незалежних малих сервісів, що взаємодіють через HTTP, WebSockets або AMQP. Кожен мікросервіс має власну логіку, життєвий цикл і може розгортатися автономно. Переваги: гнучкість, масштабованість і легке супроводження великих систем.

3. **Хід роботи**

Тема : CI server (state, command, decorator, mediator, visitor, soa)

Архітектура системи "CI Server" (SOA Implementation)

У рамках курсової роботи було спроєктовано та реалізовано систему на базі Сервіс-Орієнтованої Архітектури (SOA). Система складається з двох автономних вузлів, які взаємодіють між собою через мережу за допомогою протоколу HTTP (REST API).

Система розділена на три логічні та фізичні компоненти:

1. Service Provider (Серверна частина - CiServer.Web):

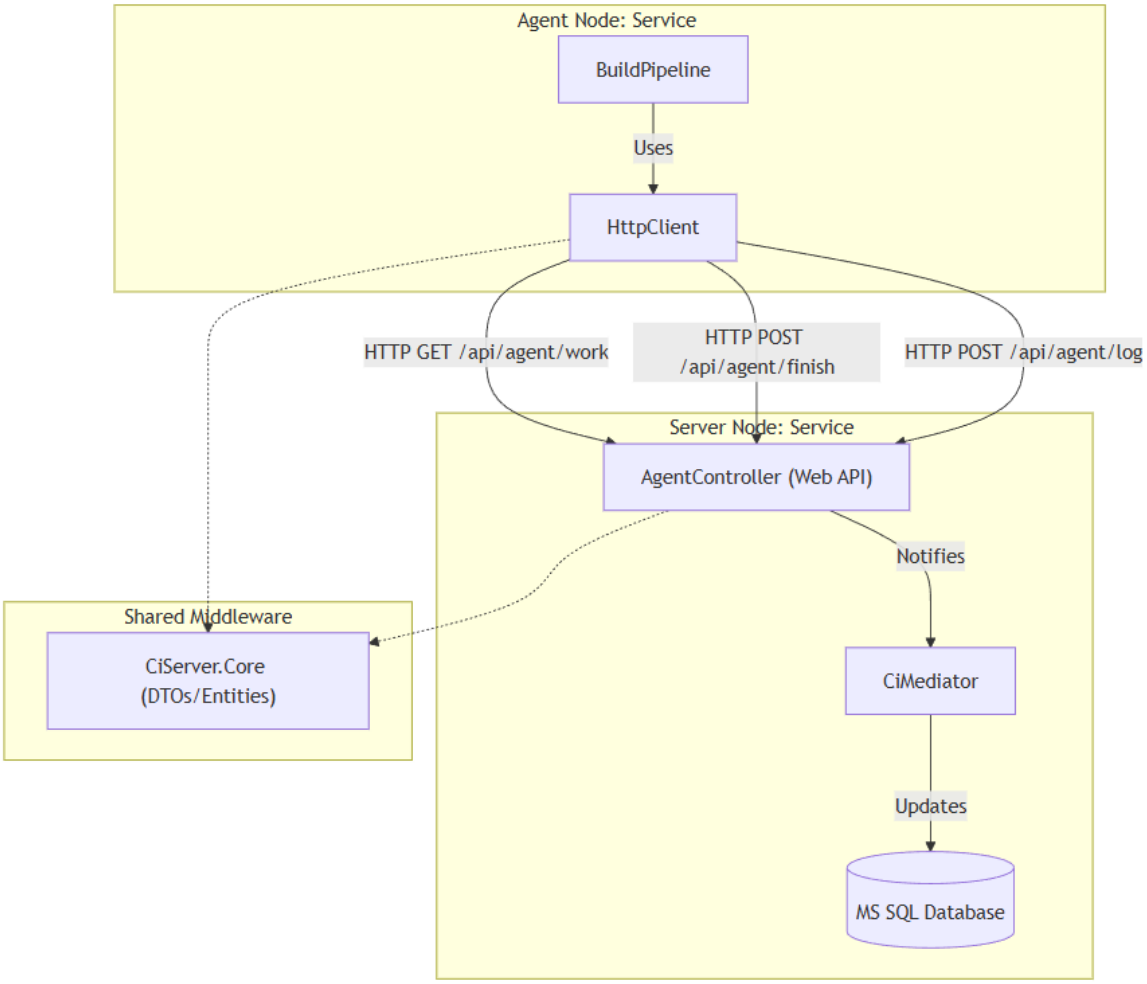
- Виступає як центральний веб-сервіс (Web API).
- Відповідає за оркестрацію процесів, збереження даних у БД (MS SQL) та надання інтерфейсу користувача.
- Публікує API-ендпоінти (/api/agent/...) для зовнішніх споживачів.

2. Service Consumer (Клієнтська частина - CiServer.Agent):

- Автономний консольний додаток, який може бути запущений на будь-якій машині в мережі.
- Не має прямого доступу до бази даних.
- Виконує "брудну роботу" (компіляція, тестування) і звітує перед Сервісом.

3. Data Contracts (Загальна частина - CiServer.Core):

- Бібліотека класів, що містить спільні сутності (Build, Project, BuildLog).
- Виконує роль контракту даних, гарантуючи, що Сервер і Агент "розуміють" один одного при серіалізації/десеріалізації JSON.



Реалізація взаємодії (Code Snippets)

Взаємодія реалізована за принципом Polling (Опитування). Агент періодично звертається до Сервісу, запитуючи нові задачі. Це дозволяє Агенту знаходитися за NAT або брандмауером без необхідності відкривати вхідні порти.

1. Серверна частина (Service Definition - AgentController.cs): Цей клас визначає контракт сервісу. Він приймає запити та повертає дані у форматі JSON.

```
namespace CiServer.Web.Controllers;

[Route("api/[controller]")]
[ApiController]
1 reference
public class AgentController : ControllerBase
{
    3 references
    private readonly ApplicationDbContext _context;
    3 references
    private readonly IMediator _mediator;

    0 references
    public AgentController(ApplicationDbContext context, IMediator mediator)
    {
        _context = context;
        _mediator = mediator;
    }

    [HttpGet("work")]
    0 references
    public async Task<IActionResult> GetWork()
    {
        var build = await _context.Builds
            .Include(b => b.Project)
            .FirstOrDefaultAsync(b => b.Status == BuildStatus.Pending);

        if (build == null)
            return NoContent();

        build.RestoreState();
        build.Start();
        await _context.SaveChangesAsync();

        Console.WriteLine($"[SERVER] Assigned job {build.BuildId}");
        return Ok(build);
    }
```

```
[HttpPost("finish")]
0 references
public IActionResult FinishWork([FromBody] Build result)
{
    _mediator.Notify(this, "JobFinished", result);
    return Ok();
}

[HttpPost("log")]
0 references
public IActionResult AddLog([FromBody] BuildLog log)
{
    _mediator.Notify(this, "LogReceived", log);
    return Ok();
}
```

2. Клієнтська частина: Агент використовує стандартний HttpClient для комунікації. Він не знає про внутрішню будову сервера, лише про API.

```
const string SERVER_URL = "http://localhost:5086";
using var httpClient = new HttpClient { BaseAddress = new Uri(SERVER_URL) };

Console.WriteLine($"--- CI AGENT CONNECTED TO {SERVER_URL} ---");

while (true)
{
    try
    {
        var response = await httpClient.GetAsync("/api/agent/work");

        if (response.IsSuccessStatusCode && response.StatusCode != System.Net.HttpStatusCode.NoContent)
        {
            var build = await response.Content.ReadFromJsonAsync<Build>();
            if (build != null)
            {
                await ExecutePipelineAsync(build, httpClient);
            }
        }
        else
        {
            Console.Write(".");
            await Task.Delay(2000);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"[ERROR] Server unavailable: {ex.Message}");
        await Task.Delay(5000);
    }
}
```

3. Відправка логів у реальному часі (Decorator + Network): Агент відправляє проміжні результати, використовуючи той самий HttpClient.

```
private void SendLog(string content)
{
    var log = new BuildLog { BuildId = _buildId, Content = content };
    _client.PostAsJsonAsync("/api/agent/log", log);
    Console.WriteLine($"   -> Sent log: {content}");
}
```

Результати роботи

Система успішно демонструє розподілену роботу. Сервер та Агент були запущені як окремі процеси.

```
[JOB STARTED] BuildId: 59a74616-1f33-4e1d-bb85-8fbc23c1deb5
>>> Pipeline started.
   -> Sent log: [Start] CloneRepositoryCommand
[GIT] Preparing workspace: C:\Users\38093\AppData\Local\Temp\CiBuilds\59a74616-1f33-4e1d-bb85-8fbc23c1deb5
[GIT] Cloning https://github.com/gfnf9977/aishki...
   ! Cloning into '.'...
[GIT] Repository cloned successfully.
   -> Sent log: [Success] CloneRepositoryCommand
   -> Sent log: [Start] CompileCodeCommand
[COMPILER] Building project in C:\Users\38093\AppData\Local\Temp\CiBuilds\59a74616-1f33-4e1d-bb85-8fbc23c1deb5...
   > MSBUILD : error MSB1003: укажите проект или файл решения. Текущий рабочий каталог не содержит проект или файл решения.
   -> Sent log: [Error] CompileCodeCommand: Compilation failed.
!!! Pipeline failed on command HttpReportDecorator: Compilation failed.
[JOB FINISHED] Status: Failed
```

Консоль Агента: Показує процес отримання задачі від сервера, виконання команд (Git, Build) та відправку звітів назад.

Project Report: 7654

Repository: https://github.com/gfnf9977/aishki

Build History:

- **Build ID:** 59a74616-1f33-4e1d-bb85-8fbc23c1deb5
  Status: Failed
  Time: 08.12.2025 15:01:08

```
[15:01:10] [Start] CloneRepositoryCommand
[15:01:12] [Success] CloneRepositoryCommand
[15:01:12] [Start] CompileCodeCommand
[15:01:12] [Error] CompileCodeCommand: Compilation failed.
```

Веб-інтерфейс Сервера: Показує, що дані, надіслані Агентом, були успішно збережені в БД і відображені користувачу.

4. Висновок

У ході виконання лабораторної роботи було реалізовано систему на базі Service-Oriented Architecture (SOA). Розроблений додаток складається з двох незалежних частин:

1. ASP.NET Core Web API (Сервер), який виступає провайдером послуг та точкою доступу до даних.
2. Console Worker (Агент), який є споживачем послуг і виконує ресурсомісткі задачі на віддаленому вузлі.

Взаємодія між компонентами забезпечується через протокол HTTP (REST architectural style), а узгодженість даних гарантується використанням спільної бібліотеки CiServer.Core. Така архітектура дозволяє легко масштабувати систему, додаючи нові екземпляри агентів на різні фізичні машини без зміни коду сервера.

5. Контрольні питання

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель організації комп'ютерних систем, де клієнти (зазвичай користувацькі програми або пристрої) роблять запити на сервер, який обробляє ці запити і повертає результати. Клієнт відповідає за інтерфейс користувача та ініціацію запитів, сервер — за обробку даних, зберігання та логіку.

2. Розкажіть про сервіс-орієнтовану архітектуру (SOA).

SOA — це архітектурний підхід, де функціональність програми реалізована у вигляді сервісів — автономних компонентів, що виконують конкретні бізнесзавдання. Кожен сервіс має чіткий інтерфейс і взаємодіє з іншими через стандартизовані протоколи (наприклад, HTTP, SOAP, REST).

3. Якими принципами керується SOA?

Основні принципи SOA:

• Автономність сервісів — сервіси працюють незалежно.

• Стандартизовані інтерфейси — сервіси взаємодіють через визначені API.

• Повторне використання — сервіси можна використовувати в різних системах.

• Легко інтегрувати — сервіси повинні легко поєднуватись у складні процеси.

• Слабке зв'язування (loose coupling) — зміни в одному сервісі мінімально

впливають на інші.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють через стандартизовані повідомлення або API, наприклад через SOAP або REST. Кожен сервіс публікує свій інтерфейс (WSDL, OpenAPI), і інші сервіси можуть викликати його методи, обмінюючись даними у формі XML, JSON або інших форматах.

5. Як розробники дізнаються про існуючі сервіси і як робити до них запити?

• Реєстр сервісів (Service Registry) — централізована база, де зареєстровані всі сервіси та їхні інтерфейси.

• Документація API (наприклад OpenAPI/Swagger).

• Запити здійснюються через стандартні протоколи (HTTP, SOAP, REST) за адресою сервісу і з використанням описаних методів та форматів даних.

5. У чому полягають переваги та недоліки клієнт-серверної моделі?

Переваги:

• Центральне зберігання даних, легший контроль безпеки.

• Легко масштабувати сервери.

• Клієнти можуть бути простими, вся логіка на сервері.

Недоліки:

• Сервер може стати вузьким місцем (single point of failure).

• Високі вимоги до потужності сервера при великій кількості клієнтів.

• Залежність клієнта від сервера — без доступу до сервера система не працює.

7. У чому полягають переваги та недоліки однорангової (peer-to-peer) моделі взаємодії?

Переваги:

• Відсутність централізованого сервера, підвищена стійкість до відмов.

• Можливість прямого обміну ресурсами між учасниками.

• Масштабування «горизонтальне» — додаючи вузли, підвищуєш потужність.

Недоліки:

• Складніше забезпечувати безпеку та контроль доступу.

• Кожен вузол відповідає за управління ресурсами.

• Важче координувати оновлення і синхронізацію даних.

8. Що таке мікросервісна архітектура? Мікросервісна архітектура — це підхід, де додаток складається з малих, незалежних сервісів, кожен з яких реалізує одну бізнес-функцію і може розгортатися окремо. Вона є розвитком SOA, але з більш дрібними і автономними компонентами.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?

• HTTP/HTTPS + REST

• gRPC

• SOAP

• Message brokers: RabbitMQ, Kafka, MQTT для асинхронної взаємодії

• WebSockets для реального часу

10. Чи можна назвати підхід сервіс-орієнтованою архітектурою, якщо у проєкті між веб-контролерами та шаром доступу до даних реалізуємо шар бізнес-логіки у вигляді сервісів?

Це не повноцінна SOA, а локальне використання сервісів всередині одного додатку. SOA передбачає автономні, незалежні сервіси, доступні через стандартизовані протоколи для інших систем. Твій підхід — це архітектурний патерн «сервісний шар» (Service Layer), який організовує бізнес-логіку всередині одного проєкту.