



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

Лабораторна робота № 9

із дисципліни «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Варіант - 17

Виконав

Студент групи IA-31:

Лисенко В. Є

Перевірив:

Мягкий М. Ю.

Київ 2025

Зміст

1. Мета:	3
2. Теоретичні відомості:	3
3. Завдання:	3
4. Хід роботи:	3
Програмна реалізація:	6
5. Висновок	9
6. Контрольні питання:	10

1. Мета:

Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented 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), спільній (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, блокчайн, 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. Завдання:

- 1) Ознайомитись з короткими теоретичними відомостями.
- 2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- 3) Реалізувати один з розглянутих шаблонів за обраною темою.
- 4) Реалізувати не менше 3-х класів відповідно до обраної теми.
- 5) Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

4. Хід роботи:

Мій варіант:

17. **System activity monitor** (iterator, command, abstract factory, bridge, visitor, SOA) Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

Для реалізації взаємодії компонентів системи було обрано Сервіс-орієнтований підхід (Service-Based Architecture) в рамках настільного застосунку. Система розділена на три логічні рівні (Layers), що забезпечує слабку зв'язність (Loose Coupling) та чіткий розподіл відповідальності:

1. Presentation Layer (Шар представлення): Відповідає за взаємодію з користувачем (MainWindow, UserWindow). Цей шар виступає в ролі "Клієнта". Він не має прямого доступу до бази даних і не містить бізнес-логіки.
2. Service Layer (Шар сервісів): Містить бізнес-логіку програми (SystemController, AnalysisVisitor). Цей шар виступає в ролі "Сервісу". Він обробляє команди від клієнта (генерація, очищенння, аналіз) та керує даними.
3. Data Access Layer (Шар даних): Відповідає за збереження стану системи (MonitorDbContext, Entity Framework).

Така архітектура дозволяє в майбутньому легко масштабувати систему до розподіленої (Distributed SOA), замінивши прямі виклики методів контролера на HTTP-запити до веб-сервісу без змін у логіці обробки даних.

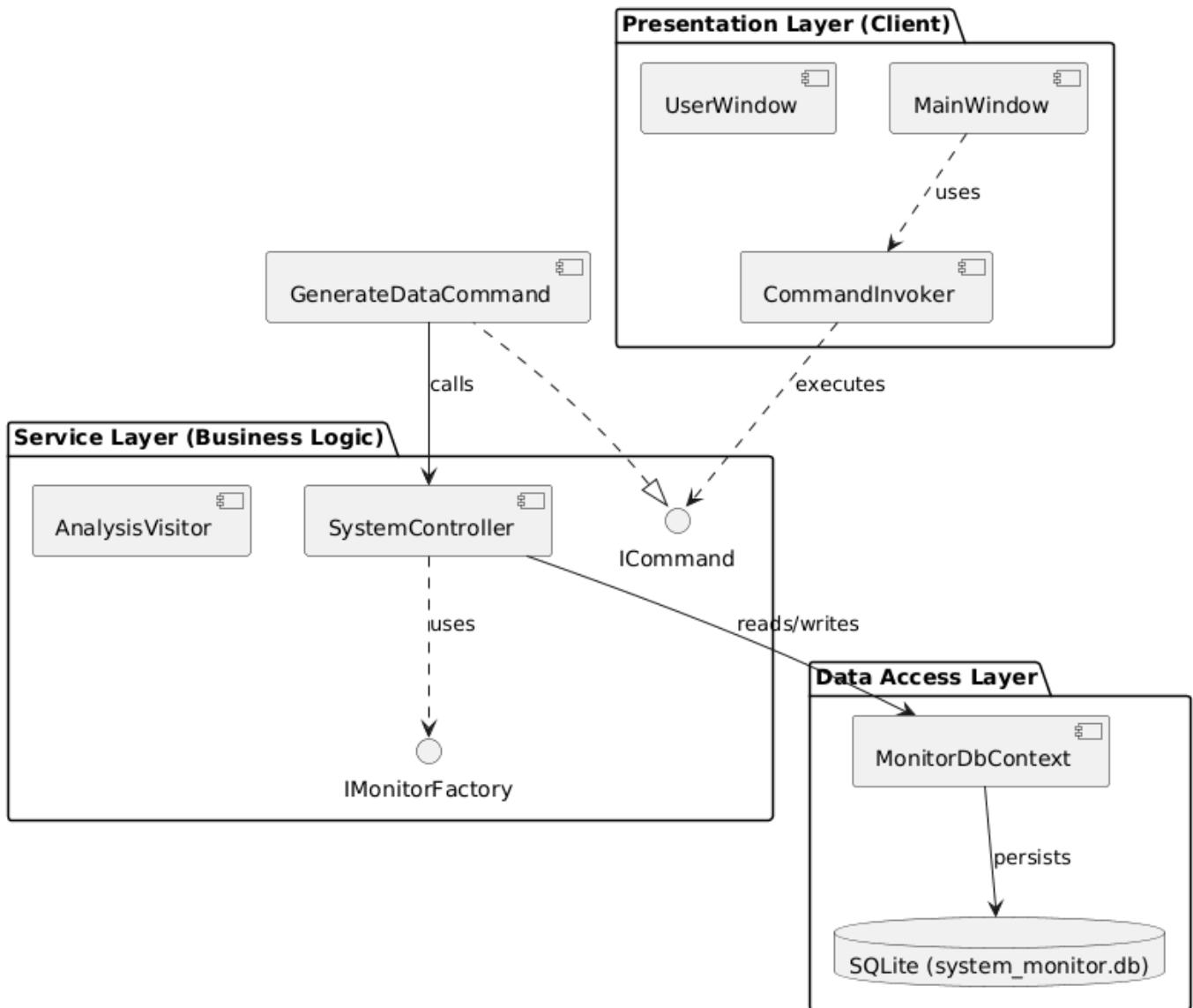


Рисунок 1 - Діаграма компонентів

Система побудована за принципом розділення на шари:

- **Presentation Layer:** `MainWindow` ініціює дії через патерн `Command`. Це забезпечує ізоляцію UI від логіки.
- **Service Layer:** Центральним компонентом є `SystemController`. Він виконує роль Фасаду/Сервісу, надаючи методи `GenerateWithDataFactory()` та `ClearAllData()`. Також сюди входить логіка аналітики (`AnalysisVisitor`).
- **Data Access Layer:** Взаємодія з базою даних інкапсульзована в `MonitorDbContext`. Використання ORM (Entity Framework) дозволяє абстрагуватися від конкретної СУБД (`SQLite`).

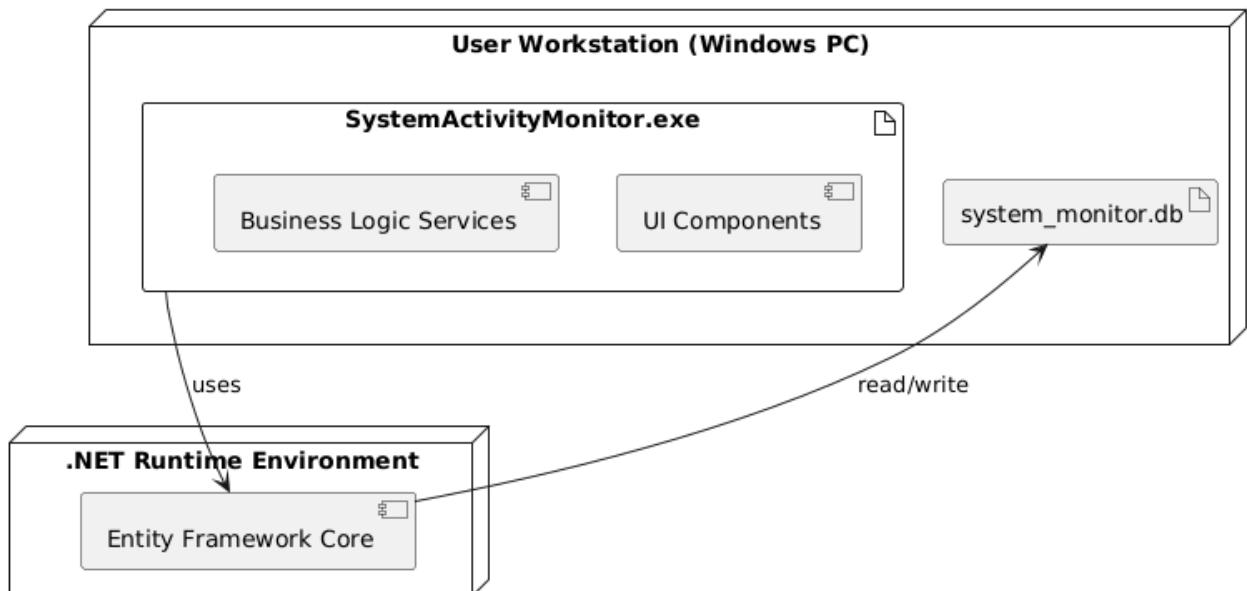


Рисунок 2 - Діаграма розгортання

Діаграма розгортання (рис. 2) демонструє фізичну архітектуру системи «System Activity Monitor» та розподіл програмних артефактів по обчислювальних вузлах. Система побудована як настільний застосунок (Desktop Application), що розгортається на Робочій станції користувача (User Workstation) під управлінням ОС Windows.

Основними елементами діаграми є:

1. Середовище виконання (.NET Runtime Environment): Необхідна платформа для запуску керованого коду C#.
2. Артефакт SystemActivityMonitor.exe: Головний виконуваний файл програми, який містить логіку представлення (UI) та бізнес-сервіси.
3. Компонент Entity Framework Core: ORM-бібліотека, що забезпечує рівень доступу до даних та абстрагує роботу з SQL.
4. Артефакт system_monitor.db: Локальний файл бази даних SQLite, який виступає сховищем для логів активності, облікових записів користувачів та налаштувань.

Взаємодія між застосунком та базою даних відбувається в межах одного фізичного вузла через внутрішні механізми вводу-виводу файлової системи (In-Process Communication).

Програмна реалізація:

Клієнтська частина (MainWindow.xaml.cs):

```

private void BtnGenerate_Click(object sender, RoutedEventArgs e)
{
    IMonitorFactory selectedFactory;

    if (cmbFactoryMode.SelectedIndex == 0)
        selectedFactory = new StandardFactory();
}

```

```

    else
        selectedFactory = new CriticalFactory();

    ICommand generateCmd = new GenerateDataCommand(_controller, selectedFactory);

    _invoker.SetCommand(generateCmd);
    _invoker.Run();

    MessageBox.Show($"Дані згенеровано! Режим: {((ComboBoxItem)cmbFactoryMode.SelectedItem).Content}");
    BtnLoadIterator_Click(null, null);
}

```

Сервісна частина (SystemController.cs):

```

public void GenerateDataWithFactory(IMonitorFactory factory)
{
    ICpuSensor cpuSensor = factory.CreateCpuSensor();
    IRamSensor ramSensor = factory.CreateRamSensor();

    using (var db = new MonitorDbContext())
    {
        var admin = db.Users.FirstOrDefault(u => u.Username == "admin");
        if (admin == null) return;

        var session = new Session
        {
            UserId = admin.Id,
            MachineName = "FULL-PC",
            OSVersion = "Windows 11 Pro"
        };
        db.Sessions.Add(session);
        db.SaveChanges();
        var rnd = new Random();
        string[] windows = { "Google Chrome", "Visual Studio", "Google Chrome", "Telegram", "Word" };

        for (int i = 0; i < 10; i++)
        {
            db.ResourceLogs.Add(new ResourceLog
            {
                SessionId = session.Id,
                CpuLoad = cpuSensor.GetCpuLoad(),
                RamUsage = ramSensor.GetFreeRam(),
                ActiveWindow = windows[rnd.Next(windows.Length)],
                CreatedAt = DateTime.UtcNow.AddSeconds(i * 2)
            });
        }
    }
}

```

```

    });

}

string[] keyActions = { "Ctrl+C", "Ctrl+V", "Enter", "Alt+Tab", "Space" };

string[] mouseActions = { "Left Click", "Right Click", "Scroll Down", "Double Click" };

for (int i = 0; i < 5; i++)
{
    bool isKeyboard = rnd.Next(0, 2) == 0;

    db.InputEvents.Add(new InputEvent
    {
        SessionId = session.Id,
        EventType = isKeyboard ? "Keyboard" : "Mouse",
        Details = isKeyboard
            ? keyActions[rnd.Next(keyActions.Length)]
            : mouseActions[rnd.Next(mouseActions.Length)],
        CreatedAt = DateTime.UtcNow.AddSeconds(i * 3)
    });
}

db.SaveChanges();
}
}

```

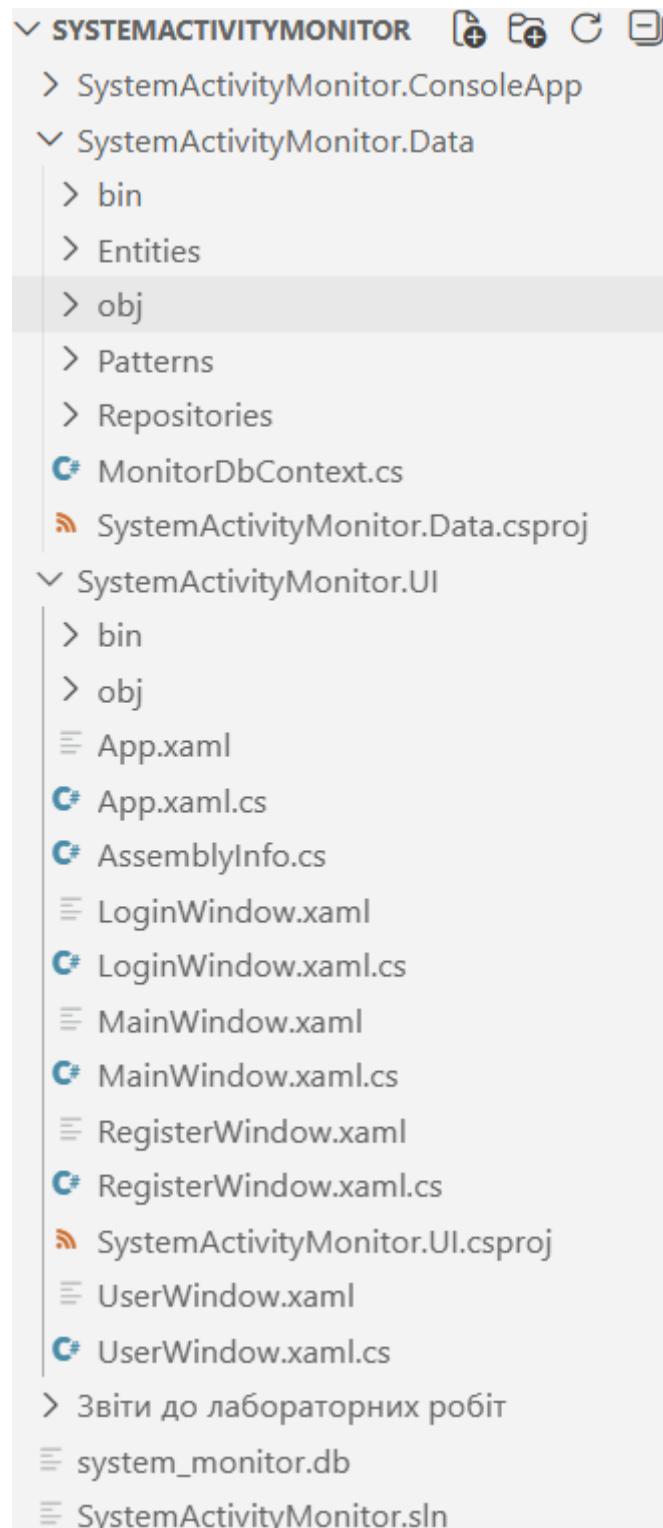


Рисунок 3 – Структура проекту, що відображає поділ на рівні

- SystemActivityMonitor.UI (Presentation)
- SystemActivityMonitor.Data -> Patterns (Business Logic)
- SystemActivityMonitor.Data -> MonitorDbContext (Data Access)

5. Висновок

У ході виконання лабораторної роботи було проаналізовано різні види архітектури програмних систем (Client-Server, P2P, SOA). Для системи моніторингу активності було реалізовано сервіс-орієнтований підхід на рівні архітектури застосунку (Service-Based Application Architecture).Хоча фізично система розгорнута як монолітний Desktop-додаток, логічно вона розділена на три незалежні шари: Представлення (UI), Бізнес-сервіси (Logic) та Дані (Data Access). Роль сервісу виконує клас SystemController, який інкапсулює бізнес-правила та роботу з фабриками даних. Взаємодія між UI та Сервісом реалізована через патерн Command, що забезпечує слабку зв'язність компонентів. Така архітектура повністю відповідає принципам SOA (чіткі межі сервісів, автономність логіки), але реалізована в рамках одного процесу для спрощення розгортання. Це створює надійну основу для можливого майбутнього перенесення логіки SystemController у веб-сервіс (ASP.NET Web API) без необхідності переписувати клієнтську частину.

6. 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 проекті 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.