Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 9

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи.»

Виконала:
студентка групи ІА-31
Кизим Є. К.

Київ 2025

Зміст

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

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
- Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.
- Реалізувати взаємодію розподілених частин:
  - *Для клієнт-серверних варіантів*: реалізація клієнтської і серверної частини додатків, а також загальної частини (middleware); зв'язок клієнтської і серверної частин за допомогою WCF, TcpClient, .NET Remoting на розсуд виконавця.
  - *Для однорангових мереж*: реалізація взаємодії клієнтських додатків за допомогою WCF Peer to peer channel.
  - *Для SOA додатків*: реалізація сервісу, що надає послуги клієнтським застосуванням; викладання сервісу в хмару або підняття у вигляді Web Service на локальній машині; використання токенів для передачі даних про автентифікації, двостороннє шифрування.
- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє спроєктовану архітектуру. Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури.

Моя тема: 27. Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA) Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних;

різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

Теоретичні відомості

Клієнт-серверна архітектура — це модель взаємодії, що розділяє систему на дві сторони: Сервер (постачальник ресурсів) та Клієнт (споживач). Клієнт завжди ініціює взаємодію, надсилаючи запит, а сервер обробляє його та повертає результат.

Основні переваги такої моделі:

- Централізований контроль і безпека даних.
- Простота оновлення (зміни лише на сервері).
- Менші вимоги до потужності клієнтських пристроїв, оскільки "важкі" обчислення можуть виконуватися на сервері.

У даній роботі для реалізації взаємодії розподілених частин (клієнтської і серверної) використовуються сокети та класи TcpClient / TcpListener з простору імен .NET System.Net.Sockets. Обмін даними відбувається через мережевий потік (NetworkStream), де інформація передається у вигляді масиву байтів.

Хід роботи

Діаграма взаємодії класів Клієнт-Сервер (TCP)

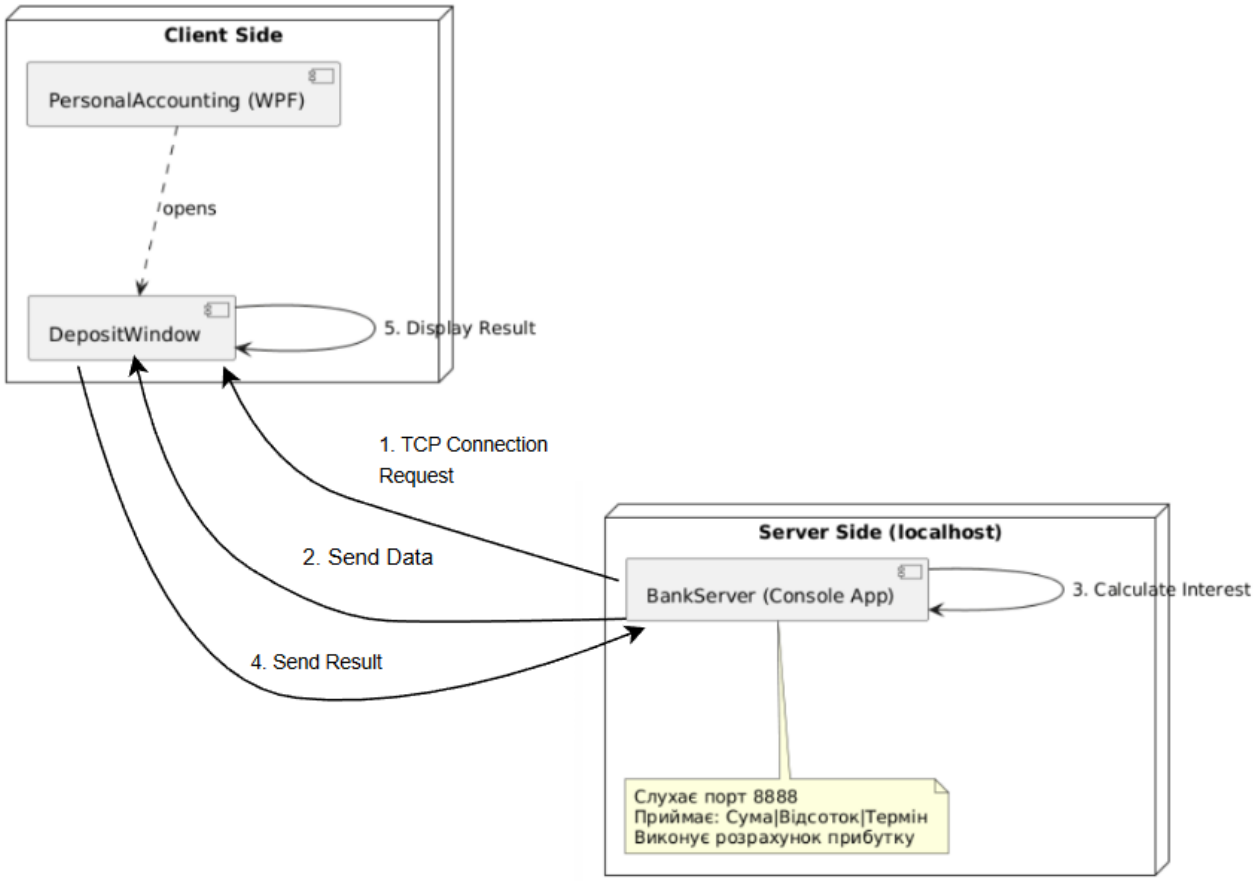


Рисунок 1 – Діаграма взаємодії класів Клієнт-Сервер (TCP)

Опис реалізованої архітектури:

1. **Серверна частина (BankServer):**
    - Реалізована як консольний додаток.
    - Використовує TcpListener для прослуховування порту 8888 на локальній адресі 127.0.0.1.
    - Відповідає за бізнес-логіку розрахунку складних відсотків по депозиту. Приймає вхідні дані у текстовому форматі, розпарсує їх, виконує обчислення та відправляє результат назад клієнту.
2. **Клієнтська частина (PersonalAccounting):**

- Реалізована як WPF-додаток (вікно DepositWindow).
- Використовує TcpClient для встановлення з'єднання із сервером.
- Відповідає за інтерфейс користувача: збір даних (сума, ставка, термін) та відображення отриманого від сервера результату.

Такий підхід дозволяє винести логіку розрахунків за межі основного додатку, імітуючи роботу з реальним банківським API.

Фрагменти програмного коду:

Серверна частина (BankServer/Program.cs) :

```
using System;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace BankServer
{
   class Program
   {
      const int PORT = 8888;

      static void Main(string[] args)
      {
         TcpListener server = null;
         try
         {
            IPAddress localAddr = IPAddress.Parse("127.0.0.1");
            server = new TcpListener(localAddr, PORT);
```

```
    server.Start();

    Console.WriteLine($"Bank Server started on port {PORT}...");
    Console.WriteLine("Waiting for requests from 'Personal Accounting'...");

    while (true)
    {
        using (TcpClient client = server.AcceptTcpClient())
        using (NetworkStream stream = client.GetStream())
        {
            Console.WriteLine("Client connected!");

            byte[] buffer = new byte[256];
            int bytesRead = stream.Read(buffer, 0, buffer.Length);
            string data = Encoding.UTF8.GetString(buffer, 0, bytesRead);

            Console.WriteLine($"Data received: {data}");

            string response = CalculateDeposit(data);

            byte[] responseData = Encoding.UTF8.GetBytes(response);
            stream.Write(responseData, 0, responseData.Length);

            Console.WriteLine($"Result sent: {response}");
        }
    }
}
catch (Exception e)
{
    Console.WriteLine($"Error: {e.Message}");
```

```
        }
        finally
        {
            server?.Stop();
        }
    }

    static string CalculateDeposit(string input)
    {
        try
        {
            string[] parts = input.Split('|');
            decimal principal = decimal.Parse(parts[0]);
            double rate = double.Parse(parts[1]);
            int months = int.Parse(parts[2]);

            decimal profit = principal * (decimal)(rate / 100) * months / 12;
            decimal total = principal + profit;

            return $"{total:F2}";
        }
        catch
        {
            return "Error";
        }
    }
  }
}
```

Клієнтська частина (PersonalAccounting/DepositWindow.xaml.cs):

```
using System;
using System.Net.Sockets;
using System.Text;
using System.Windows;

namespace personalAccounting
{
   public partial class DepositWindow : Window
   {
      private const int SERVER_PORT = 8888;
      private const string SERVER_IP = "127.0.0.1";

      public DepositWindow()
      {
         InitializeComponent();
      }

      private void Calculate_Click(object sender, RoutedEventArgs e)
      {
         try
         {
            if (string.IsNullOrWhiteSpace(AmountBox.Text) ||
               string.IsNullOrWhiteSpace(RateBox.Text) ||
               string.IsNullOrWhiteSpace(MonthsBox.Text))
            {
               MessageBox.Show("Заповніть всі поля!", "Помилка");
               return;
            }
```

```
            string message = $"{AmountBox.Text}|{RateBox.Text}|{MonthsBox.Text}";

            using (TcpClient client = new TcpClient(SERVER_IP, SERVER_PORT))
            using (NetworkStream stream = client.GetStream())
            {
                byte[] data = Encoding.UTF8.GetBytes(message);
                stream.Write(data, 0, data.Length);

                byte[] responseData = new byte[256];
                int bytes = stream.Read(responseData, 0, responseData.Length);
                string response = Encoding.UTF8.GetString(responseData, 0, bytes);

                ResultText.Text = $"{response} UAH";
            }
        }
        catch (SocketException)
        {
            MessageBox.Show("Не вдалося підключитися до Банківського
сервера.\nПереконайтеся, що програма 'BankServer' запущена!", "Помилка
мережі");
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Помилка: {ex.Message}", "Помилка");
        }
    }
  }
}
```

Контрольні питання

1. Що таке клієнт-серверна архітектура?

Це модель, що розділяє систему на дві сторони: Сервер (постачальник ресурсів) та Клієнт (споживач). Клієнт завжди ініціює взаємодію, надсилаючи запит, а сервер обробляє його та повертає результат.

2. Розкажіть про сервіс-орієнтовану архітектуру.

SOA (Service-Oriented Architecture) — це стиль проєктування, де додаток складається з окремих, слабо пов'язаних модулів (сервісів), що спілкуються через стандартні протоколи. Це гнучка альтернатива жорсткій монолітній архітектурі.

3. Якими принципами керується SOA?

Основні принципи: слабка зв'язність (мінімальна залежність між компонентами), використання універсальних інтерфейсів для комунікації та можливість повторного використання (наприклад, огортання старих систем у нові сервіси).

4. Як між собою взаємодіють сервіси в SOA?

Взаємодія відбувається виключно через обмін повідомленнями (HTTP, SOAP, REST). Ключове правило: кожен сервіс керує власною базою даних і не має прямого доступу до даних інших сервісів («share-nothing»).

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

Сервіси публікуються у спеціальному реєстрі, доступному для розробників. Маршрутизація та обмін даними часто проходять через центральну магістраль — Enterprise Service Bus (ESB).

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

Переваги: Централізований контроль і безпека даних, простота оновлення (зміни лише на сервері), менші вимоги до потужності клієнтських пристроїв.

Недоліки: Ризик повної зупинки системи при падінні сервера (Single Point of Failure), можливі перевантаження сервера та висока вартість його обслуговування.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?
Переваги: Висока живучість (немає єдиного центру), легке масштабування за рахунок ресурсів нових учасників.
Недоліки: Складність адміністрування, ризики безпеки, нестабільна доступність даних (вузли можуть зникати).

8. Що таке мікро-сервісна архітектура?
Це еволюція SOA на базі сучасних технологій. Система будується як сукупність невеликих, повністю автономних сервісів, кожен з яких відповідає виключно за одну бізнес-функцію і може розгортатися незалежно.

9. Які протоколи використовуються для обміну даними в мікросервісній архітектурі?
HTTP/REST: для синхронних запитів.
gRPC: для швидкої міжсервісної взаємодії.
AMQP/Черги повідомлень: для асинхронного фонового зв'язку.

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?
Ні, це приклад багатошарової архітектури (Layered Monolith). У цьому випадку «сервіси» — це лише класи в оперативній пам'яті однієї програми. Справжня SOA вимагає, щоб компоненти були розподіленими та спілкувалися через мережу.

Висновки: Під час виконання лабораторної роботи я ознайомилась із принципами побудови розподілених програмних систем та дослідила архітектуру «Клієнт-Сервер».

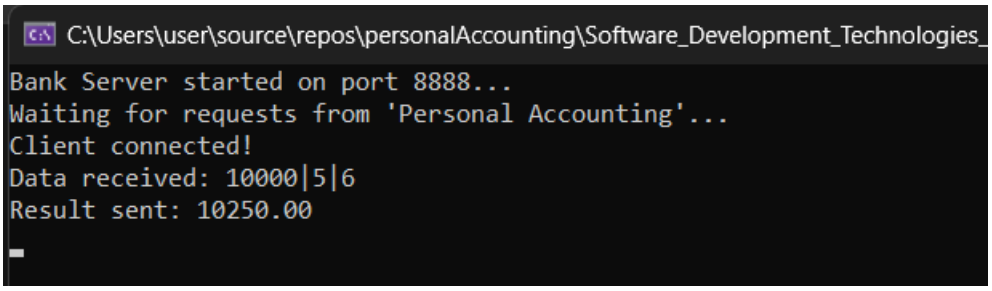
Практична частина роботи полягала в реалізації мережевої взаємодії між компонентами системи «Особиста бухгалтерія». Для виконання завдання щодо контролю банківських вкладів та звірки відсотків було розроблено окремий серверний додаток BankServer (Console App) та клієнтський модуль DepositWindow (WPF). У результаті було створено працездатну розподілену систему, де основна програма виступає в ролі клієнта, що звертається до сервера за фінансовими розрахунками, що відповідає вимогам до побудови масштабованих та модульних систем.

Репозиторій: посилання

ДОДАТКИ

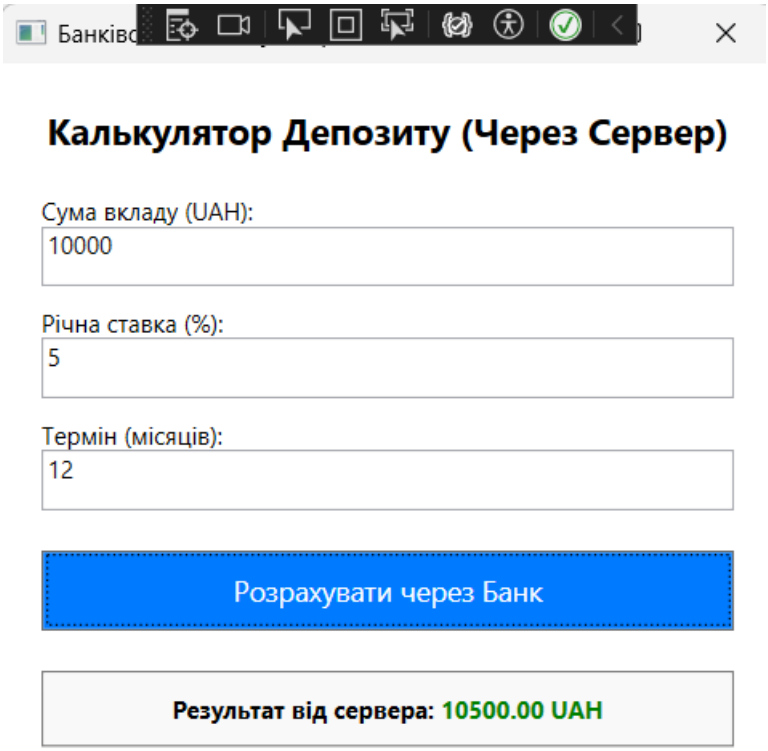
Додаток А

Інтерфейс серверного додатку BankServer, що демонструє логування подій: підключення клієнта, отримання вхідних даних та відправку розрахованого результату



Додаток Б

Вікно «Банківський Калькулятор» у клієнтському додатку, яке відправляє запит на сервер та відображає отриману суму депозиту.


Банківс
Калькулятор Депозиту (Через Сервер)
Сума вкладу (UAH):
10000
Річна ставка (%):
5
Термін (місяців):
12
Розрахувати через Банк
Результат від сервера: 10500.00 UAH