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 «Теорія розробки прогрманого забезпечення»

Тема: «Патерни проектування.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Вступ

Тема: Патерни проектування.

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

Тема проєкту: Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server). Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Теоретичні відомості

Клієнт-серверна архітектура

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти.

Тонкий клієнт – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки.

У такому варіанті використання майже все навантаження лягає на сервер або групу серверів.

Перевагою таких моделей є простота розгортання, тому що оновлювати потрібно лише сервери і в результаті клієнти з наступними запитами автоматично будуть працювати з оновленою системою.

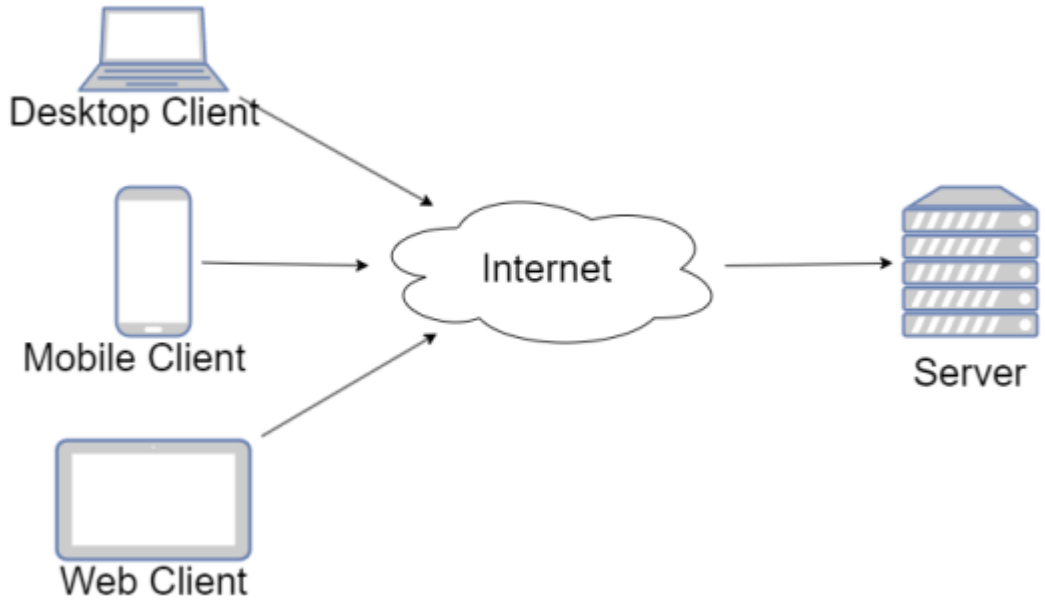

Рисунок 9.1. Клієнт-серверна архітектура

Товстий клієнт – антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких випадках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими комп'ютерами. Перевагою такого підходу є менші вимоги до серверної частини. Також перевагою, при певному підході до реалізації, є можливість працювати клієнтам без тимчасового доступу до серверу. Прикладом товстого клієнта можна назвати мобільні застосунки, або десктоп застосунки. Наприклад, Evernote, Viber, MS Outlook, комп’ютерні антивіруси, ігри, що потребують інсталяції (The Sims, GTA, …) та інші.

Проміжним варіантом можна назвати SPA (Single Page Application) – це товсті Web-клієнти, які при старті кожен раз завантажуються з сервера, а надалі працюють з сервером через web-API. З одного боку більшу частину логіки вони відпрацьовують на клієнтський стороні, за рахунок чого зменшується серверне навантаження. Також оновлення простіше ніж для товстих клієнтів. Але такі застосунки не працюють, якщо сервер не доступний.

Клієнт-серверна взаємодія, як правило, організовується за допомогою 3-х рівневої структури: клієнтська частина, загальна частина, серверна частина.

Оскільки велика частина даних загальна (класи, використовувані системою), їх прийнято виносити в загальну частину (middleware) системи.

Клієнтська частина містить візуальне відображення і логіку обробки дії користувача; код для встановлення сеансу зв'язку з сервером і виконання відповідних викликів.

Серверна частина містить основну логіку роботи програми (бізнес-логіку) або ту її частину, яка відповідає зберіганню або обміну даними між клієнтом і сервером або клієнтами.

Хід роботи

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, .NETRemoting 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.

Реалізація Cient-Server архітектури

Для реалізації веб-орієнтованого терміналу використано архітектуру **Клієнт-Сервер (Client-Server)**, де чітко розмежовано функції представлення та обробки даних. Веб-браузер виступає у ролі «тонкого» клієнта, що відповідає за візуалізацію інтерфейсу, тоді як серверна частина (Back-end) бере на себе виконання всієї бізнес-логіки. Обмін даними здійснюється через REST API з використанням асинхронних запитів (AJAX), що дозволяє відправляти команди та отримувати результати без повного перезавантаження сторінки.

Архітектура реалізована через взаємодію REST-контролера (TerminalApiController) на стороні сервера та JavaScript-клієнта на стороні браузера. Клієнт формує JSON-запити з командами, а сервер обробляє їх (використовуючи патерни *Command*, *Bridge* та *Interpreter*) і повертає готовий для відображення результат. Такий підхід є ефективним для веб-застосунків, оскільки централізує логіку виконання скриптів на сервері та забезпечує швидку реакцію інтерфейсу.

```java
package org.kpi.dto;

import lombok.Data;

@Data  2 usages  new *
public class CommandRequest {
    private String commandText;
}
```

Рис. 1 – Код класу CommandRequest

```java
package org.kpi.dto;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data  3 usages  new *
@AllArgsConstructor
public class CommandResponse {
    private String resultHtml;
}
```

Рис. 2 – Код класу CommandResponse

```java
@RestController  new *
@RequestMapping("/api")
public class TerminalApiController {

    private final PowerShellService powerShellService;  2 usages
    private final CommandInvoker commandInvoker;  2 usages

    Rename usages
    public TerminalApiController(PowerShellService powerShellService, CommandInvoker commandInvoker) {  new *
        this.powerShellService = powerShellService;
        this.commandInvoker = commandInvoker;
    }

    @PostMapping("/execute")  new *
    public CommandResponse executeCommand(@RequestBody CommandRequest request) {
        PowerShellExecuteCommand command = new PowerShellExecuteCommand(powerShellService, request.getCommandText());

        // Виконання через Invoker
        String resultHtml = commandInvoker.executeCommand(command);

        return new CommandResponse(resultHtml);
    }
}
```

Рис. 3 – Код класу TerminalApiController

Застосування цієї архітектури забезпечує:

- Чітке розмежування обов'язків: Клієнтська частина (Front-end) відповідає виключно за візуалізацію інтерфейсу та взаємодію з користувачем, тоді як серверна частина (Back-end) концентрує в собі всю бізнес-логіку, роботу з базою даних та безпосереднє виконання PowerShell-скриптів.
- Централізоване управління логікою: Сервер виступає єдиною точкою входу для виконання команд через TerminalApiController. Це дозволяє легко оновлювати алгоритми обробки команд або виправляти помилки в одному місці (на сервері), автоматично поширюючи зміни для всіх клієнтів.
- Легкість доступу («Тонкий клієнт»): Оскільки ресурсоємні операції виконуються на сервері, клієнтський пристрій не потребує потужного апаратного забезпечення або встановлення додаткового ПЗ. Доступ до терміналу можливий з будь-якого пристрою, що має веб-браузер.
- Інтерактивність та швидкість: Використання асинхронних запитів (AJAX) дозволяє інтерфейсу залишатися чутливим і не блокуватися під час виконання команд на сервері. Сторінка не перезавантажується повністю, оновлюється лише область історії команд.
- Стандартизація взаємодії: Використання протоколу HTTP та формату даних JSON створює універсальний інтерфейс (REST API). Це дозволяє в майбутньому легко розширювати систему, наприклад, додаючи мобільний клієнт, без необхідності змінювати серверну частину.

У нашому випадку Client-Server архітектура дозволяє створити веб-орієнтований термінал, де система функціонує наступним чином:

- Клієнт ініціює запити на виконання команд, формуючи JSON-об'єкти.
- Сервер приймає запити, обробляє їх через патерни Command/Bridge та взаємодіє з операційною системою.
- Інтерфейс динамічно оновлюється на основі отриманих відповідей без перезавантаження сторінки.
- Система підтримує одночасну роботу кількох клієнтів, ізолюючи їхні сесії на рівні сервера.

Зображення архітектури Client-Server

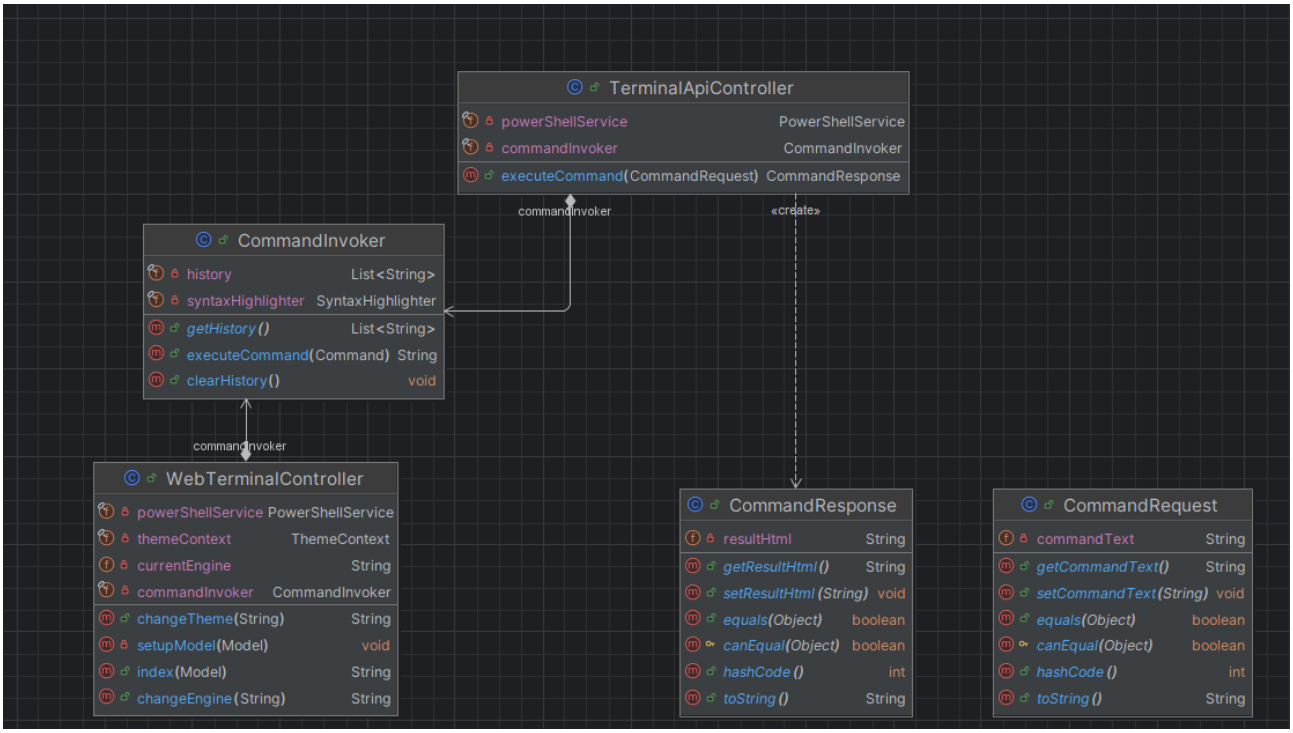

Рис. 3 – Структура архітектури Client-Server

Питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти.

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA, англ. service-oriented architecture) – модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов’язаних (англ. Loose coupling) сервісів або служб, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами.

3. Якими принципами керується SOA?

Модульність та розподіленість: Це модульний підхід до розробки, заснований на використанні розподілених сервісів (служб).

Слабка пов'язаність (Loose coupling): Сервіси є слабо пов'язаними між собою.

Стандартизація інтерфейсів: Сервіси оснащені стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами (як правило, SOAP або REST).

Взаємодія через повідомлення: Сервіси взаємодіють виключно шляхом обміну повідомленнями.

Ізоляція даних: Взаємодія відбувається без створення спеціальних інтеграцій для доступу до спільних даних (наприклад, до однієї бази даних).

Реєстрація та виявлення: Сервіси реєструються у спеціальних каталогах (сервісах), що дозволяє будь-якій команді розробників знаходити їх та використовувати.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють між собою тільки за рахунок обміну повідомленнями, без створення спеціальних інтеграцій для доступу до однієї інформації, наприклад, до однієї бази даних.

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

Згідно SOA сервіси реєструються на спеціальних сервісах і будь-яка команда розробників, якій потрібен доступ може знайти їх та використовувати.

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

+ Простота розгортання та оновлення (для моделі «Тонкий клієнт»)

+ Зменшення навантаження на сервер (для моделі «Товстий клієнт»)

+ Можливість автономної роботи (для моделі «Товстий клієнт»)

+ Простота архітектури

– Єдина точка відмови:

– Залежність від центрального вузла:

– Високе навантаження на сервер (для моделі «Тонкий клієнт»):

7. У чому полягають переваги та недоліки однорангової моделі взаємодії?

+ Децентралізація – відсутність центрального сервера, що зменшує залежність від одного вузла, підвищуючи стійкість мережі до збоїв і атак.

+ Рівноправність вузлів – кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси).

+ Розподіл ресурсів – вузли надають доступ до своїх власних ресурсів, таких як обчислювальна потужність, дисковий простір або файли.

– До основних проблемних зон можна віднести безпеку, синхронізацію даних та пошук ресурсів

8. Що таке мікро-сервісна архітектура?

Мікро-сервісна архітектура є підходом до створення серверного додатку як набору малих служб. Це означає, що архітектура мікро-сервісів головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в

своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP. Кожен мікросервіс реалізує специфічні можливості в предметній області і свою бізнес-логіку в рамках конкретного обмеженого контексту, повинна розроблятись автономно і розвертатись незалежно.

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

HTTP/HTTPS, WebSockets чи AMQP

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, цей підхід сам по собі не можна назвати сервіс-орієнтованою архітектурою (SOA) у тому розумінні, як вона визначена.

Хоча виділення шару бізнес-логіки у "сервіси" є гарною практикою структурування коду, це не перетворює архітектуру на SOA, доки ці сервіси не стануть окремими, незалежними компонентами, що взаємодіють через мережу.

Висновок

Під час виконання лабораторної роботи я вивчив види взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізував в проєктованій системі Power Shell термінал архітектуру Client-Server.