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

Лабораторна робота №9
Технології розроблення програмного забезпечення
«Взаємодія компонентів системи.»

Тема: Редактор зображень

Виконала:

студентка групи ІА-32

Ейсмонт А.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Тема: Взаємодія компонентів системи.

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

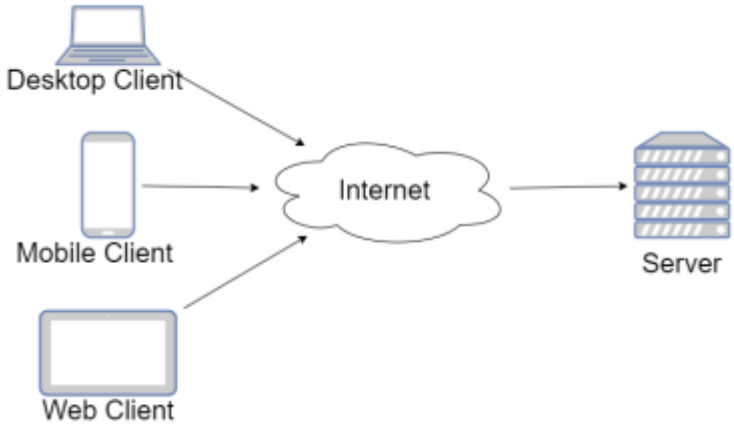
Завдання:

- 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, .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.

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.



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.

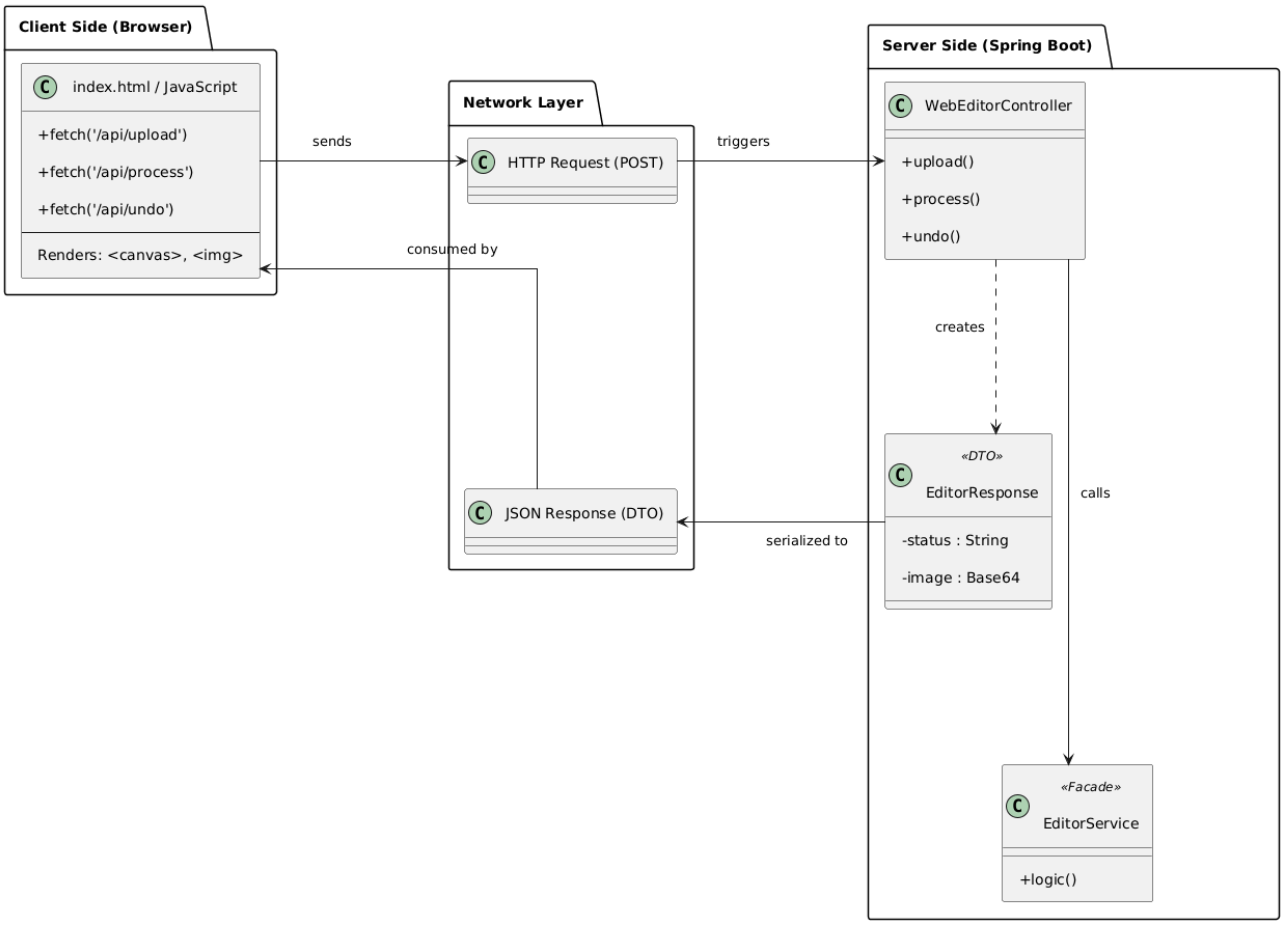
Хід роботи:

Опис спроєктованої архітектури (Client-Server):
Було реалізовано розподілену систему на базі архітектури **«Клієнт — Сервер»**. Взаємодія між компонентами відбувається за протоколом HTTP з використанням архітектурного стилю REST.

Компоненти системи:

1. **Клієнтська частина (Frontend):**
   - Реалізована як Single Page Application (SPA) на базі HTML5, CSS3 та JavaScript.
   - Відповідає виключно за візуалізацію інтерфейсу, обробку подій миші (малювання на Canvas) та відправку асинхронних запитів на сервер (AJAX/Fetch API).
   - Не містить бізнес-логіки обробки зображень.
2. **Серверна частина (Backend):**
   - Реалізована на платформі Java Spring Boot.
   - Приймає HTTP-запити, виконує обробку зображень (фільтри, композиція шарів), керує станом та історією, зберігає дані в БД.
   - Виступає як API-сервер, повертаючи дані у форматі JSON.
3. **Протокол взаємодії (Communication):**
   - Для обміну даними використовуються **DTO (Data Transfer Objects)**. Це дозволяє чітко структурувати відповіді сервера (статус, дані зображення, помилки) і відокремити внутрішню модель даних (Entity) від зовнішнього інтерфейсу.

Діаграма класів:



Фрагменти програмного коду

Data Transfer Object: Клас, який визначає формат відповіді сервера клієнту.

EditorResponse.java

```
public class EditorResponse {
    private String status;
    private String image;       // Base64 encoded image
    private String errorMessage;

    public EditorResponse(String status, String image) {
        this.status = status;
        this.image = image;
    }
    // Getters & Setters...
}
```

Серверна частина (Controller): Обробка запиту та формування DTO-відповіді.

WebEditorController.java

```
@RestController
public class WebEditorController {

    @PostMapping("/api/process/{action}")
    public ResponseEntity<EditorResponse> process(@PathVariable String action, HttpSession session) {
        try {
            // Виклик бізнес-логіки
            String resultBase64 = editorService.applyFilter(session.getId(), action);

            // Повернення DTO (JSON)
            return ResponseEntity.ok(new EditorResponse("Success", resultBase64));
        } catch (Exception e) {
            return ResponseEntity.status(403).body(new EditorResponse("Error", e.getMessage(), true));
        }
    }
}
```

Клієнтська частина: Асинхронний запит до сервера та обробка отриманого DTO.

Index.html

```
async function applyFilter(type) {
    const res = await fetch(`/api/process/${type}`, { method: 'POST' });

    if (res.ok) {
        const data = await res.json();

        console.log(data.status); // "Success"
```

```
        bgImg.src = "" + data.image;
    }
}
```

Питання до лабораторної роботи:

1. Що таке клієнт-серверна архітектура?

Клієнт-серверні додатки — це найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовуються для зберігання і обробки даних). Клієнт-серверна взаємодія, як правило, організовується за допомогою 3-х рівневої структури: клієнтська частина, загальна частина (middleware), серверна частина.

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA) — це модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов'язаних сервісів або служб, оснащених стандартизованими інтерфейсами для взаємодії за стандартизованими протоколами. Історично вона з'явилася як альтернатива монолітній архітектурі. Програмні комплекси в SOA зазвичай реалізуються як набір веб-служб.

3. Якими принципами керується SOA?

SOA керується принципами використання розподілених, слабо пов'язаних сервісів зі стандартизованими інтерфейсами. Сервіси взаємодіють між собою тільки за рахунок обміну повідомленнями, без створення спеціальних інтеграцій для доступу до однієї інформації (наприклад, до однієї бази даних).

4. Як між собою взаємодіють сервіси в SOA?

Сервіси, як правило, взаємодіють по HTTP з використанням SOAP або REST. Взаємодія відбувається виключно шляхом обміну повідомленнями. Часто реалізація покладається на використання централізованого компонента — шини даних (Enterprise Service Bus).

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

Згідно з 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?

- *Тонкий клієнт: Перевагою є простота розгортання, оскільки оновлювати потрібно лише сервери. Недоліком є те, що майже все навантаження лягає на сервер.*

– *Товстий клієнт: Перевагами є менші вимоги до серверної частини та можливість працювати без тимчасового доступу до сервера, оскільки більшість логіки міститься на стороні клієнта.*

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

– *Переваги: Децентралізація (відсутність центрального сервера підвищує стійкість), рівноправність вузлів та розподіл ресурсів (вузли надають власні ресурси).*
– *Недоліки: Проблеми з безпекою (складно контролювати дані), складність синхронізації даних та зниження ефективності пошуку ресурсів зі збільшенням кількості вузлів.*

8. Що таке мікро-сервісна архітектура?

Це підхід до створення серверного додатку як набору малих служб. Кожен мікросервіс реалізує специфічні можливості в предметній області, має свою бізнес-логіку, розробляється автономно і розгортається незалежно.

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

Служби взаємодіють за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP. Взаємодія підтримується за допомогою зв'язку на основі повідомлень.

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 не можна вважати повноцінною сервіс-орієнтованою архітектурою (SOA), оскільки тут відбувається підміна понять. У нашому випадку слово «сервіс» означає просто клас у коді, який містить бізнес-логіку і викликається контролером напряму через звичайний виклик методу в межах однієї запущеної програми. Справжня ж SOA передбачає, що сервіси — це окремі незалежні програми, які часто працюють на різних серверах і спілкуються між собою через мережу за допомогою спеціальних протоколів, а не просто через виклик функцій у коді.

Вихідний код:
https://github.com/eismonta/TRPZ_2025/tree/e2702494ea37db026df6b4c509ef277ab1bdb1df/image-editor

Висновок:
Протягом виконання лабораторної роботи я ознайомилась з видами взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізувала взаємодію компонентів системи за архітектурою **Клієнт-Сервер**.