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

із дисципліни ««Технології розроблення програмного забезпечення»
Тема: «Взаємодія компонентів системи»

Виконав:
студент групи ІА-31:
Кунда А.П.

Перевірив:
Мягкий М.Ю

Київ 2025

Тема: Патерни проектування

Тема проєкту: 28. JSON Tool (ENG) (strategy, command, observer, template method,flyweight) Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Autosave\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view.

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

Посилання на репозиторій з проєктом:
https://github.com/Octopus663/json-tool

Хід роботи

1) Ознайомитись з короткими теоретичними відомостями

Клієнт-серверна архітектура Клієнт-серверна архітектура — це модель, у якій клієнт відповідає за взаємодію з користувачем, а сервер — за зберігання й обробку даних. Тонкий клієнт передає більшість операцій на сервер і лише відображає результати (наприклад, вебзастосунки). Його перевага — просте оновлення, адже зміни виконуються лише на сервері. Товстий клієнт виконує більшу частину логіки на своїй стороні, що зменшує навантаження на сервер і дозволяє працювати офлайн (мобільні та десктопні застосунки — Evernote, Viber, Outlook тощо). SPA (Single Page Application) — проміжний варіант: логіка виконується на клієнті, але робота можлива лише при наявності зв’язку з сервером. Типова структура клієнт-серверної системи має три рівні: клієнтський (інтерфейс і взаємодія), спільний (middleware) і серверний (бізнес-логіка та робота з даними).

Peer-to-Peer архітектура P2P — децентралізована модель, у якій кожен вузол одночасно є клієнтом і сервером. Усі учасники рівноправні й обмінюються ресурсами без центрального сервера. Основні принципи: децентралізація, рівноправність вузлів і розподіл ресурсів. Приклади: BitTorrent, блокчейн, Skype, Zoom, розподілені обчислення (BOINC). 3 Недоліки: складність забезпечення безпеки, синхронізації й пошуку даних у великих мережах.

Сервіс-орієнтована архітектура (SOA) SOA — модульний підхід до створення системи як набору незалежних сервісів зі стандартизованими інтерфейсами, що взаємодіють через HTTP, SOAP або REST. Сервіси виконують конкретні бізнес-функції й обмінюються повідомленнями, не маючи спільної бази даних. Можуть бути обгортками для старих систем і реєструються в сервісному каталозі. Часто використовують Enterprise Service Bus (ESB) для взаємодії між сервісами. SOA стала основою для розвитку мікросервісної архітектури.

Мікросервісна архітектура Мікросервісна архітектура — це створення додатків як набору незалежних малих сервісів, що взаємодіють через HTTP, WebSockets або AMQP. Кожен мікросервіс має власну бізнес-логіку, життєвий цикл і може розгортатися автономно. За визначенням з книги О'Рейлі, мікросервіс — це незалежний компонент із чіткими межами, що спілкується через повідомлення. Переваги: гнучкість, масштабованість і легке супроводження великих систем.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.
4. Реалізувати взаємодію розподілених частин.

Оскільки ми вже використовуємо Spring Boot, який ідеально підходить для створення REST API, найлогічнішим і найсучаснішим варіантом буде реалізація SOA (Service-Oriented Architecture) у вигляді RESTful Web Service.

Ми реалізуємо архітектуру, де:

- Сервер: Надає "послуги" (Services) через чіткий інтерфейс (API). Він не займається малюванням HTML-сторінок, а повертає чисті дані (JSON).
- Клієнт (Frontend): Споживає ці послуги.

2. Реалізація REST API

Нам потрібно створити окремий контролер, який буде повертати чистий JSON.

Ми створимо DocumentRestController, який буде мати методи для отримання та створення документів.

3. Реалізація Клієнтської взаємодії (Middleware/Client)

Для Spring Boot стандартом є використання RestTemplate або WebClient для спілкування між сервісами.

4. Автентифікація та Токени (Security)

Додамо Spring Security і реалізуємо захист, щоб до нашого API не міг звернутися будь-хто.

Створення REST-контролера

```java
package com.example.jsontool.controller;

import com.example.jsontool.model.JSONDocument;
import com.example.jsontool.service.DocumentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;


@RestController  new *
@RequestMapping("/api/documents")
public class DocumentRestController {

    @Autowired
    private DocumentService documentService;

    @GetMapping  new *
    public List<JSONDocument> getAllDocuments() {
        return documentService.findAll();
    }

    @GetMapping("/{id}")  new *
    public ResponseEntity<JSONDocument> getDocumentById(@PathVariable Long id) {
        return documentService.findAll().stream()  Stream<JSONDocument>
                .filter( JSONDocument d -> d.getId().equals(id))
                .findFirst()  Optional<JSONDocument>
                .map(ResponseEntity::ok)  Optional<ResponseEntity<...>>
                .orElse(ResponseEntity.notFound().build());
    }

    @PostMapping  new *
    public ResponseEntity<JSONDocument> createDocument(@RequestBody JSONDocument document) {
        try {
            documentService.save(document);
            return ResponseEntity.ok(document);
        } catch (Exception e) {
            return ResponseEntity.badRequest().build();
        }
    }
}
```

DocumentRestController.java

Додавання Spring Security

```xml
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
    </dependencies>
```

pom.xml

```java
package com.example.jsontool.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

import static org.springframework.security.config.Customizer.withDefaults;

@Configuration  new *
@EnableWebSecurity
public class SecurityConfig {

    @Bean  new *
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf( CsrfConfigurer<HttpSecurity> csrf -> csrf.disable())
                .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
                        .requestMatchers("/", "/documents/**", "/css/**", "/js/**").permitAll()
                        .requestMatchers("/api/**").authenticated()
                )
                .httpBasic(withDefaults());

        return http.build();
    }

    @Bean  new *
    public UserDetailsService userDetailsService() {
        UserDetails user = User.withDefaultPasswordEncoder()
                .username("admin")
                .password("password")
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(user);
    }
}
```

SecurityConfig.java

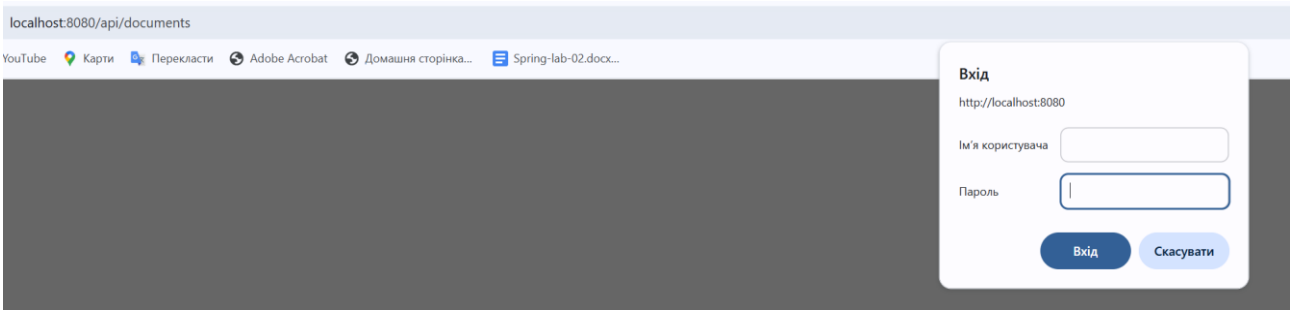
Реалізація "Клієнта"

```java
package com.example.jsontool.client;

import org.springframework.web.client.RestTemplate;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;

import java.util.Base64;


public class JsonToolClient {  new *

    private static final String API_URL = "http://localhost:8080/api/documents";  1 usage

    public static void main(String[] args) {  new *

        RestTemplate restTemplate = new RestTemplate();

        String plainCreds = "admin:password";
        byte[] plainCredsBytes = plainCreds.getBytes();
        String base64Creds = Base64.getEncoder().encodeToString(plainCredsBytes);
        HttpHeaders headers = new HttpHeaders();
        headers.add( headerName: "Authorization",  headerValue: "Basic " + base64Creds);

        HttpEntity<String> request = new HttpEntity<>(headers);

        try {
            System.out.println("--- КЛІЄНТ: Відправляю запит до Сервісу... ---");
            ResponseEntity<String> response = restTemplate.exchange(
                    API_URL,
                    HttpMethod.GET,
                    request,
                    String.class
            );

            System.out.println("--- СЕРВІС ВІДПОВІВ: ---");
            System.out.println("Статус: " + response.getStatusCode());
            System.out.println("Дані (JSON): " + response.getBody());

        } catch (Exception e) {
            System.out.println("Помилка доступу: " + e.getMessage());
        }
    }
}
```

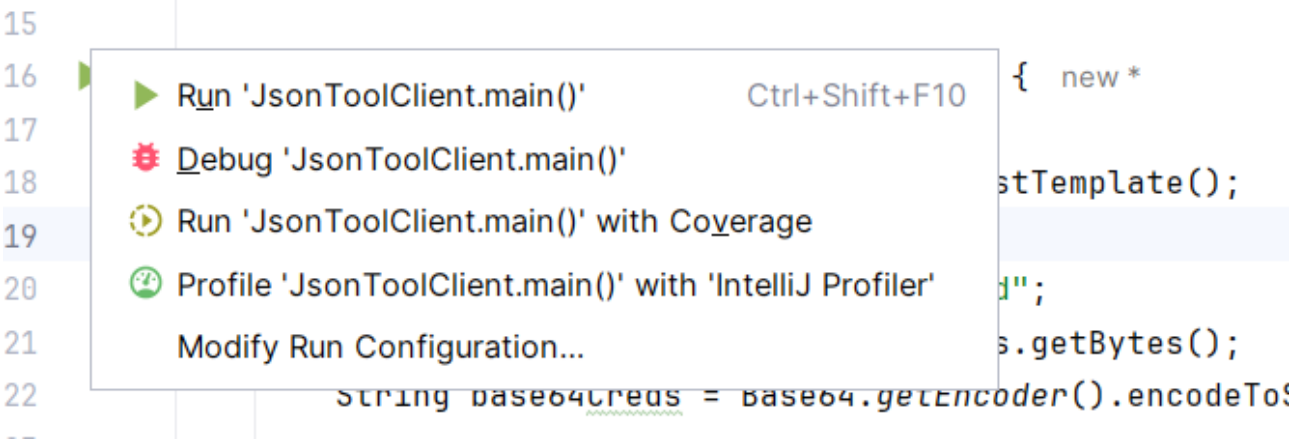
JsonToolClient.java

Демонастрація роботи

Якщо ми тепер спробуємо зайти по посиланню, яка веде до документів, то побачимо вікно входу, яке не пускає нас далі.



Тепер запустимо наш JsonToolClient



Бачимо, які дані він видає:

```
C:\Users\PaulK\.jdks\openjdk-21.0.2\bin\java.exe ...
--- КЛІЄНТ: Відправляю запит до Сервісу... ---
--- СЕРВІС ВІДПОВІВ: ---
Статус: 200 OK
Дані (JSON): [{"id":1,"name":"Burger","content":"{\r\n  Name:\"Paul\",\r\n  Age: 15,\r\n  Sex:

Process finished with exit code 0
```

Як висновок, у консолі ми бачимо, як клієнт успішно звернувся до сервера, передав токен, і сервер віддав йому список документів у форматі JSON.

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 шаблон

Висновки

Під час виконання лабораторної ми вивчили принципи роботи сервіс-орієнтованої архітектури (SOA) та реалізували її у своєму проєкті. Метою було розділити систему на незалежні компоненти та організувати їх взаємодію через REST API. Для цього було створено REST API, безпеку за допомогою Spring Security, і також створено клієнт. Таким чином, проєкт було трансформовано з монолітного веб-додатку в систему, що підтримує SOA, де сервер надає послуги, а клієнти можуть споживати їх через стандартизований інтерфейс.

Відповіді на контрольні питання

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель взаємодії в комп'ютерних мережах, де завдання розподіляються між постачальниками послуг (серверами) та замовниками послуг (клієнтами). Клієнт ініціює спілкування, надсилаючи запит серверу, а сервер обробляє запит і надсилає відповідь.

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

SOA (Service-Oriented Architecture) — це архітектурний стиль, в якому додатки будуються з окремих, незалежних та слабко пов'язаних компонентів, що називаються сервісами. Ці сервіси надають бізнес-функціонал через чітко визначені інтерфейси (контракти) і можуть спілкуватися між собою через мережу.

3. Якими принципами керується SOA?

Основні принципи SOA:

- Слабка зв'язаність (Loose Coupling): Сервіси мінімально залежать один від одного.
- Стандартизований контракт: Сервіси взаємодіють через чітко визначені інтерфейси (WSDL, Swagger/OpenAPI).
- Автономність: Кожен сервіс контролює свою власну логіку та дані.
- Повторне використання: Сервіси розробляються так, щоб їх можна було використовувати в різних додатках.
- Відсутність стану (Statelessness): Сервіси не повинні зберігати інформацію про стан клієнта між запитами.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють через механізм обміну повідомленнями. Це може бути:

- SOAP (Simple Object Access Protocol): Важковаговий протокол на основі XML.
- REST (Representational State Transfer): Легковаговий архітектурний стиль, що використовує HTTP (GET, POST, PUT, DELETE) та JSON/XML.
- Черги повідомлень (Message Queues): Асинхронна взаємодія (наприклад, RabbitMQ, Kafka).

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

- Реєстр сервісів (Service Registry): Централізована база даних, де зберігається інформація про доступні сервіси та їхні адреси (наприклад, UDDI в SOAP або Eureka в мікросервісах).
- Опис інтерфейсу: Розробники використовують документи опису API (наприклад, WSDL для SOAP або Swagger/OpenAPI для REST), щоб зрозуміти, які методи доступні, які параметри потрібні та який формат відповіді очікувати.

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

- Переваги: Централізація даних (легше керувати та захищати), простота обслуговування (оновлення тільки на сервері), масштабованість сервера.
- Недоліки: Єдина точка відмови (якщо сервер впаде, ніхто не працює), перевантаження сервера при великій кількості клієнтів, висока вартість обладнання для сервера.

7. У чому полягають переваги та недоліки однорангової моделі взаємодії (P2P)?

- Переваги: Висока відмовостійкість (немає центрального сервера), легка масштабованість (кожен новий вузол додає ресурси), низька вартість (використовуються ресурси користувачів).
- Недоліки: Складність керування та адміністрування, проблеми з безпекою, відсутність гарантії доступності даних (якщо вузол з даними вимкнеться).

8. Що таке мікро-сервісна архітектура?

Мікросервісна архітектура — це еволюція SOA, де додаток будується як набір дуже маленьких, повністю автономних сервісів, кожен з яких відповідає за одну конкретну бізнес-функцію (наприклад, сервіс "Кошик", сервіс "Оплата"). Кожен мікросервіс може бути написаний на іншій мові програмування і мати свою власну базу даних.

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

- HTTP/REST (JSON): Найпоширеніший для синхронної взаємодії.

- gRPC: Високопродуктивний протокол від Google (на основі HTTP/2 та Protobuf).
- AMQP/MQTT: Протоколи для асинхронного обміну повідомленнями через брокери (RabbitMQ, Kafka).

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?

Технічно, ні, це не SOA. Це просто шарувата (Layered) архітектура монолітного додатку.

- У моноліті "сервіс" — це просто Java-клас (наприклад, DocumentService), який викликається іншим класом в тій самій пам'яті JVM.
- У SOA "сервіс" — це окрема програма, що працює незалежно і спілкується з іншими через мережу. Тобто, просто назвати клас "Service" не робить архітектуру сервіс-орієнтованою.