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

Тема проєкту ‘Shell (total commander)’

Виконала: студентка групи ІА-33

Маляревич Анжела Валентинівна

Перевірив:

Мягкий Михайло Юрійович

Київ - 2025

Зміст

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

Теоретичні відомості

1. Клієнт-серверна архітектура

Система складається з:

- Клієнта — інтерфейс та запити користувача.
- Сервера — обробка логіки, зберігання даних.

Типи клієнтів:

- Тонкі — мінімум логіки на клієнті, все робить сервер.
- Товсті — багато логіки на клієнті.

Переваги: простота оновлення, централізація.
Недоліки: залежність від сервера, можливе перевантаження.

2. Peer-to-Peer (P2P)

Усі вузли рівноправні: кожен може і надсилати, і приймати дані без центрального сервера.

Переваги: децентралізація, стійкість до відмов.
Недоліки: складніше забезпечити безпеку, синхронізацію та ефективний пошук.

3. Сервіс-орієнтована архітектура (SOA)

Система складається з незалежних сервісів, які взаємодіють через стандартизовані протоколи (HTTP, SOAP, REST).

Особливості:

- слабке зв’язування сервісів;
- обмін повідомленнями;

- сервіси можна оновлювати окремо.

4. Мікросервісна архітектура

Розвиток SOA. Система складається з малих автономних сервісів, кожен — окрема функція.

Особливості:

- окремий процес для кожного сервісу;
- незалежне розгортання;
- взаємодія через HTTP/HTTPS, WebSockets, AMQP.

Переваги: масштабованість, гнучкість, просте оновлення.

Хід роботи

Діаграма класів

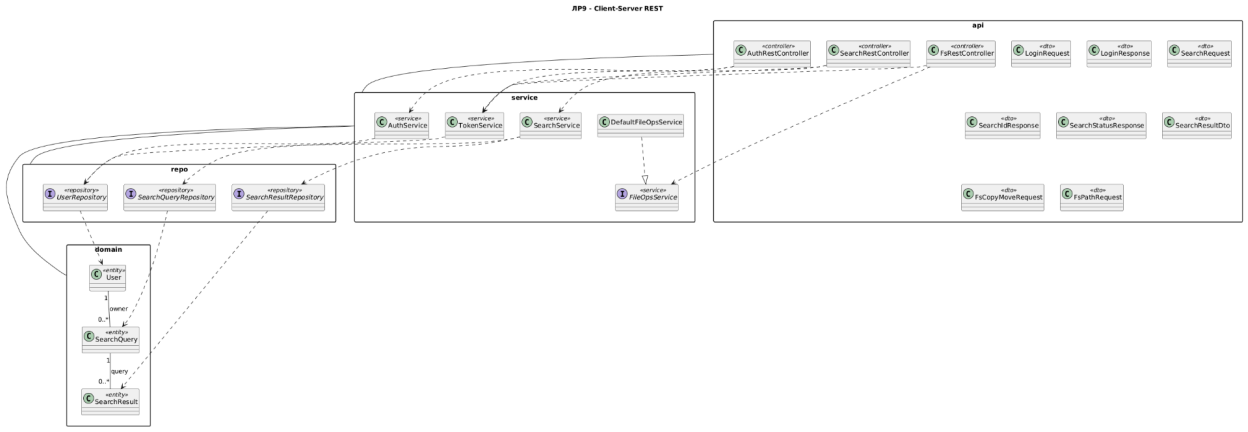

Рисунок 1 - Діаграма класів

На діаграмі зображено архітектуру REST-компонентів системи Shell File Manager. Вона складається з чотирьох основних рівнів: API-рівень (контролери), сервісний рівень (бізнес-логіка), рівень репозиторіїв та доменні сутності. Контролери відповідають за приймання HTTP-запитів клієнта та повернення JSON-відповідей. Сервіси містять основну логіку автентифікації, пошуку та файлових операцій, тоді як репозиторії забезпечують доступ до

бази даних. Таке розділення чітко демонструє клієнт–серверну модель взаємодії та спрощує масштабування і підтримку системи.

REST-логін (AuthRestController)

```
@PostMapping("/login")
public ResponseEntity<LoginResponse> login(@RequestBody LoginRequest request) {
    return authService.authenticate(request.getEmail(), request.getPassword()) Optional<User>
            .map( User user -> {
                String token = tokenService.issueTokenFor(user);
                return ResponseEntity.ok(new LoginResponse(token));
            }) Optional<ResponseEntity<...>>
            .orElseGet(() -> ResponseEntity
                    .status(HttpStatus.UNAUTHORIZED)
                    .build());
}
```

Рисунок 2 - REST-метод логування користувача та видача токена.

Після успішної автентифікації користувачу видається унікальний opaque-token. Клієнт зберігає цей токен і надсилає його в усі подальші запити до REST-API через заголовок X-Auth-Token. Таким чином реалізовано перехід від session-cookie до токенної авторизації.

Сервіс токенів (TokenService)

```
public String issueTokenFor(User user) {
    String token = UUID.randomUUID().toString();
    tokens.put(token, user.getId());
    return token;
}


public User requireUser(String token) {
    if (token == null || token.isBlank()) {
        throw new ResponseStatusException(
                HttpStatus.UNAUTHORIZED,
                "Missing authentication token"
        );
    }
```

Рисунок 3 - Генерація токена та перевірка автентичності користувача.

Пошук файлів (SearchRestController)

```
@PostMapping
public SearchIdResponse startSearch(
        @RequestHeader("X-Auth-Token") String token,
        @RequestBody SearchRequest request
) {
    User user = tokenService.requireUser(token);

    SearchQuery query = searchService.createQuery(
            user,
            request.getNameMask(),
            request.getExt()
    );

    return new SearchIdResponse(query.getId());
}
```

Рисунок 4 - REST-запуск операції пошуку файлів

Запит створює SearchQuery у базі даних та повертає клієнту унікальний ідентифікатор операції пошуку. Це відповідає підходу асинхронних REST-операцій: клієнт отримує queryId та окремо запитує статус і результати пошуку.

Операції з файлами (копіювання, переміщення, видалення)

```
@PostMapping("/copy")
public void copy(
        @RequestHeader("X-Auth-Token") String token,
        @RequestBody FsCopyMoveRequest request
) throws Exception {
    tokenService.requireUser(token);
    fileOpsService.copy(
            Path.of(request.getSrc()),
            Path.of(request.getDst()),
            int p -> {}
    );
}
```

Рисунок 5 - Виклик серверної операції копіювання файлу через REST

Файлові операції виконуються на сервері, а клієнт взаємодіє з ними через REST-кінцеві точки. Усі операції захищені токенною авторизацією.
Метод приймає два шляхи – джерело та призначення – і передає їх до FileOpsService.

Висновок

У ході роботи я реалізувала REST-API для Shell File Manager: логін, пошук та операції з файлами. Побудована UML-діаграма відобразила взаємодію між контролерами, сервісами й репозиторіями. Система отримала зручну клієнт–серверну архітектуру та токенну авторизацію.

Питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?

Це модель, у якій клієнт надсилає запити, а сервер обробляє їх і повертає результат. Клієнт містить мінімум логіки, а вся основна робота виконується на сервері.

2. Розкажіть про сервіс-орієнтовану архітектуру.

SOA — це архітектура, де система складається з незалежних сервісів, які взаємодіють між собою через стандартизовані протоколи (частіше HTTP/REST або SOAP). Сервіси слабко зв’язані, мають чіткі контракти та можуть оновлюватися незалежно.

3. Якими принципами керується SOA?

- слабке зв’язування;
- стандартизовані інтерфейси;
- повторне використання сервісів;
- обмін повідомленнями;
- незалежне розгортання компонентів.

4. Як між собою взаємодіють сервіси в SOA?

Через стандартизовані протоколи (HTTP, SOAP, REST), обмінюючись повідомленнями у чітко визначеному форматі (XML, JSON). Кожен сервіс викликає інші через опубліковані API.

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

Через документацію, опис API (OpenAPI/Swagger), або реєстр сервісів. Запити виконуються за опублікованими HTTP-ендпоінтами.

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

Переваги:

- проста структура;
- централізоване зберігання даних;
- легке оновлення серверної частини.

Недоліки:

- залежність від сервера;
- сервер може перевантажуватися;
- потрібне мережеве з’єднання.

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

Переваги:

- децентралізація;
- висока стійкість до відмов;
- відсутність центрального сервера.

Недоліки:

- складно забезпечити безпеку;
- важко синхронізувати дані;
- складний пошук вузлів.

8. Що таке мікро-сервісна архітектура?

Це підхід, у якому система складається з малих автономних сервісів, кожен з яких виконує одну чітку функцію та працює як окремий процес. Сервіси взаємодіють через HTTP/REST, WebSockets або брокери повідомлень.

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

- HTTP/HTTPS
- REST
- gRPC
- WebSockets
- AMQP (RabbitMQ)
- Kafka (stream-передача)

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?

Так, частково.
Це спрощена форма SOA всередині одного застосунку, де кожен сервіс

виконує окрему бізнес-функцію.
Але це не повна SOA, бо:

- сервісів багато, але вони не окремі процеси;
- вони не взаємодіють через мережу;
- немає незалежного розгортання.

Проте сам принцип слабкого зв’язування шарів — присутній, тому підхід близький до SOA за ідеєю.