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: «Різні види взаємодії додатків:
client-server, peer-to-peer,
service-oriented architecture»

Варіант №15

Виконав:
студент групи ІА-23
Лядський Д.С.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема

Різні види взаємодії додатків: client-server, peer-to-peer, service-oriented architecture.

Мета

Дослідити основні види взаємодії додатків, включаючи архітектури клієнт-сервер, peer-to-peer (рівний-рівному) та service-oriented architecture (архітектура, орієнтована на сервіси), їх особливості, переваги, недоліки та сфери застосування.

Короткі теоретичні відомості

Клієнт-серверні додатки

Клієнт-серверна модель взаємодії є однією з найпоширеніших у розробці розподілених систем. У цій моделі додатки поділяються на дві основні частини: клієнт і сервер.

Клієнт: забезпечує інтерфейс для взаємодії користувача із системою. Його основна функція — надсилати запити до сервера та отримувати відповіді.

Сервер: відповідає за зберігання даних, виконання бізнес-логіки та обробку запитів клієнтів.

Типи клієнтів:

Тонкий клієнт: більшість обчислювальної роботи виконується на сервері, тоді як клієнт займається лише візуалізацією отриманих даних. Це підходить для систем, де потрібно централізувати обчислення або зберігати конфіденційність даних. Однак, це створює значне навантаження на сервер та залежність від якості мережі.

Товстий клієнт: більша частина бізнес-логіки переноситься на клієнтську сторону. Це зменшує навантаження на сервер, дозволяє ефективно використовувати ресурси клієнта, але потребує додаткових ресурсів для обслуговування клієнтського додатка.

Переваги моделі:

Централізація управління даними (в тонкому клієнті).

Простота налаштування клієнтів (необхідне лише підключення до сервера).

Недоліки:

Потенційне перевантаження сервера.

Залежність продуктивності системи від якості мережі.

Peer-to-Peer (P2P)

Модель peer-to-peer (рівний-рівному) забезпечує взаємодію між рівноправними учасниками системи без використання централізованого сервера. Кожен вузол одночасно виконує роль клієнта та сервера.

Особливості:

Децентралізація: усі вузли є рівноправними і самостійно обробляють запити.

Взаємодія напряму: вузли напряму взаємодіють один з одним для обміну даними чи виконання операцій.

Проблеми:

Синхронізація даних: складність у забезпеченні актуальності даних між вузлами.

Пошук ресурсів: знайти потрібні вузли для взаємодії може бути непросто, часто використовуються структури (DHT, трекери).

Переваги:

Висока стійкість до відмов.

Відсутність єдиної точки відмови (сервер).

Гнучкість і масштабованість.

Сервіс-орієнтована архітектура (SOA)

Сервіс-орієнтована архітектура — це підхід до створення ПЗ, в якому система складається з незалежних модулів (сервісів), що взаємодіють між собою через стандартизовані інтерфейси та протоколи.

Основні принципи:

Слабка зв’язаність: сервіси незалежні один від одного, що спрощує їхню заміну чи оновлення.

Повторне використання: сервіси можуть бути використані в різних додатках.

Інкапсуляція: деталі реалізації сервісу приховані, доступ можливий лише через інтерфейси.

Переваги:

Можливість інтеграції різнорідних систем.

Підтримка хмарних рішень (SaaS, PaaS).

Масштабованість і гнучкість.

Недоліки:

Складність розробки та налаштування.

Значні витрати на підтримку стандартів.

Мікросервісна архітектура

Цей підхід до розробки додатків базується на розподілі системи на невеликі незалежні сервіси, які можуть бути розгорнуті, масштабовані та оновлені автономно.

Основні характеристики:

Малі розміри: кожен сервіс реалізує невелику частину бізнес-логіки.

Автономність: кожен сервіс функціонує окремо, що дозволяє незалежний життєвий цикл.

Комунікація через протоколи: сервіси взаємодіють через HTTP, WebSocket або черги повідомлень.

Переваги:

Легкість у масштабуванні.

Зручність у розробці великих систем.

Можливість використання різних технологій у кожному сервісі.

Недоліки:

Ускладнення комунікації між сервісами.

Потреба в оркестрації сервісів та інфраструктурі для їхньої підтримки.

Ці моделі взаємодії мають свої унікальні особливості, які визначають їхню доцільність у різних сценаріях розробки.

Завдання

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів і їх взаємодій для

досягнення конкретних функціональних можливостей.

3. Реалізувати взаємодію програми в одній з архітектур відповідно до обраної теми.

Обрана тема

15 E-mail клієнт (singleton, builder, decorator, template method, interpreter, SOA)

Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти

pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Хід роботи

Функціонал для роботи в розподіленому оточенні та взаємодія розподілених частин:

```
1 usage
public UserDto login(String username, String password) throws Exception {
    AES aes = new AES();

    UserLoginDto userLoginDto = new UserLoginDto(aes.encrypt(username), aes.encrypt(password), Base64.getEncoder().encodeToString(aes

    try {
        ResponseEntity<UserDto> response = restTemplate.postForEntity( url: userControllerUrl + "/login", userLoginDto, UserDto.class);
        UserDto enc = response.getBody();
        UserDto userDto = new UserDto(enc.getId(), aes.decrypt(enc.getUsername()), enc.getEmailAccounts(), aes.decrypt(enc.getPasswor
        System.out.println(userDto.getUsername());
        System.out.println(userDto.getJwt());
        return userDto;
    }
    catch (HttpClientErrorException e) {
        System.out.println("Error login user: " + e.getResponseBodyAsString());
        e.printStackTrace();
        throw new IllegalArgumentException("User login failed: " + e.getMessage());
    }
}
```

Рисунок 1. Код для логіну користувача в модулі представлення

```
@PostMapping("/login")
public ResponseEntity<UserDto> login(@RequestBody UserLoginDto user) {
    try {
        UserDto loginUser = userService.login(user.getUsername(), user.getPassword(), user.getKey());
        return new ResponseEntity<>(loginUser, HttpStatus.OK);
    } catch (Exception e) {
        return new ResponseEntity<>(null, HttpStatus.BAD_REQUEST);
    }
}
```

Рисунок 2. Код для логіну користувача в контролері модуля user-service

```
1 usage
@Transactional(readOnly = true)
public UserDto login(String usernameEnc, String passwordEnc, String key) throws Exception {
    byte[] decodedKey = Base64.getDecoder().decode(key);
    SecretKey secretKey = new SecretKeySpec(decodedKey,  offset: 0, decodedKey.length,  algorithm: "AES");
    String username = aes.decrypt(usernameEnc, secretKey);
    String jwt = jwtUtil.generateToken(username);
    UserDto user = userMapper.mapToUserDto(userRepository.findByUsername(username).orElseThrow(() -> new IllegalArgumentException(
    if (user.getPassword().equals(passwordEnc)) {
        return user;
    }
    throw new IllegalArgumentException("Wrong password!");
}
```

Рисунок 3. Код для логіну користувача в сервісі модуля user-service

```
11  public class AES {
12
        3 usages
13  @      public  String encrypt(String data, SecretKey key) throws Exception {
14              Cipher cipher = Cipher.getInstance( transformation: "AES");
15              cipher.init(Cipher.ENCRYPT_MODE, key);
16              byte[] encryptedBytes = cipher.doFinal(data.getBytes());
17              return Base64.getEncoder().encodeToString(encryptedBytes);
18          }
19
        4 usages
20          public  String decrypt(String encryptedData, SecretKey key) throws Exception {
21              Cipher cipher = Cipher.getInstance( transformation: "AES");
22              cipher.init(Cipher.DECRYPT_MODE, key);
23              byte[] decodedBytes = Base64.getDecoder().decode(encryptedData);
24              byte[] decryptedBytes = cipher.doFinal(decodedBytes);
25              return new String(decryptedBytes);
26          }
27  }
```

Рисунок 4. Клас для шифрування/дешифрування

```java
@Component
public class JwtUtil {
    2 usages
    private final String SECRET_KEY = "secret";

    2 usages
    public String generateToken(String username) {
        return Jwts.builder()
                .setSubject(username)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60))
                .signWith(SignatureAlgorithm.HS256, SECRET_KEY)
                .compact();
    }

    public String validateToken(String token) {
        return Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody().getSubject();
    }
}
```

Рисунок 5. Клас генерації та валідації токенів

1. Метод login у класі-клієнті

Метод login виконує:

- Шифрування імені користувача та пароля за допомогою AES.
- Створення DTO (UserLoginDto), який містить зашифровані дані користувача та секретний ключ у форматі Base64.
- Відправку HTTP-запиту типу POST до контролера /login.
- Декодування отриманих даних (UserDto) з відповіді сервера.
- У разі помилки запиту викидає виняток IllegalArgumentException.

2. Контролер /login

Контролер обробляє запит від клієнта:

- Приймає UserLoginDto, що містить зашифровані ім'я користувача, пароль і ключ.
- Викликає метод login у сервісі.
- У разі успішного виконання повертає UserDto із даними користувача та токеном JWT. Якщо сталася помилка, повертає статус BAD_REQUEST.

3. Сервіс login

Сервіс виконує:

- Декодування секретного ключа з Base64.
- Розшифрування імені користувача та пароля за допомогою AES.
- Генерацію JWT для розшифрованого імені користувача.
- Пошук користувача у базі даних за іменем.
- Порівняння зашифрованого пароля з паролем у базі.
- Повернення DTO з даними користувача та токеном JWT.

4. Компонент AES

Компонент AES надає методи:

- encrypt: Шифрує дані з використанням алгоритму AES.
- decrypt: Розшифровує зашифровані дані.
- Шифрування та розшифрування використовують секретний ключ SecretKey.

5. Компонент JwtUtil

Компонент JwtUtil

- Генерує токен JWT для імені користувача з обмеженим часом дії (1 година).
- Перевіряє валідність токена та витягує ім'я користувача з нього.

Висновок

У ході виконання курсової роботи було проведено аналіз основних типів взаємодії додатків, таких як клієнт-сервер, peer-to-peer та service-oriented architecture. Було вивчено їх архітектурні особливості, визначено ключові переваги та недоліки.

Практичним результатом роботи стала розробка моделі взаємодії додатків на основі архітектури, орієнтованої на сервіси (SOA). Реалізована модель

продемонструвала гнучкість, масштабованість та можливість повторного використання сервісів у різних компонентах системи. Це підтвердило доцільність використання SOA для розробки складних розподілених систем.