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: Особиста бухгалтерія

Виконала:
студентка групи ІА-32
Коляда М. С.

Перевірив:
Мягкий М.Ю.

Київ 2025

Зміст

Завдання..... 3

Теоретичні відомості..... 3

Діаграма класів:..... 5

Опис спроєктованої архітектури (Сервіс-орієнтована архітектура):..... 5

Фрагменти програмного коду:..... 6

Вихідний код:..... 8

Питання до лабораторної роботи..... 8

Висновок..... 11

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.

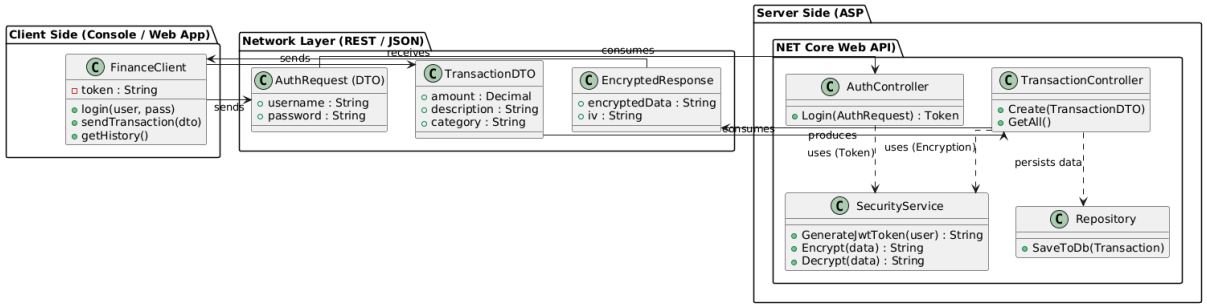
Теоретичні відомості

Сервіс-орієнтована архітектура

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. Історично сервіс-орієнтована архітектура появилася як альтернатива монолітній архітектурі, в які вся система розроблялася та розгорталася як одне ціле. Програмні комплекси, розроблені відповідно до сервіс-орієнтованою архітектурою, зазвичай реалізуються як набір веб-служб (або веб-сервісів), які, як правило, взаємодіють по HTTP з використанням SOAP або REST. Ці служби надають певні бізнес-функції, наприклад, отримання інформації про наявність матеріалів на складі. Сервіси взаємодіють між собою тільки за рахунок обміну повідомленнями, без створення спеціальних інтеграцій для доступу до однієї інформації, наприклад, до однієї бази даних. Сервіси також можуть бути реалізовані як обгортки навколо застарілої системи. Це робиться для зменшення вартості переробки системи, а також спрощення інтеграції існуючих монолітних систем в нову архітектуру. Згідно SOA сервіси реєструються на спеціальних сервісах і будь-яка команда розробників, якій потрібен доступ може знайти їх та використовувати. Часто реалізація SOA покладається на використання централізованого програмного компонента для обміну даними – шину даних (Enterprise Service Bus).

Хід роботи

Діаграма класів:



Опис спроєктованої архітектури (Сервіс-орієнтована архітектура):

Було реалізовано розподілену систему на базі Сервіс-орієнтованої архітектури (SOA). Взаємодія між компонентами відбувається за стандартизованим протоколом HTTP з використанням принципів REST, що забезпечує слабку зв'язність (Loose Coupling) між модулями.

Компоненти системи:

1. Клієнтська частина (Service Consumer):
    - Реалізована як окремий додаток (Console Application або Web Client), який виступає споживачем послуг.
    - Відповідає виключно за взаємодію з користувачем: збір вхідних даних (сума, категорія, опис транзакції) та відображення отриманих результатів (історія операцій, статус виконання).
    - Не містить бізнес-логіки розрахунків, прямого доступу до бази даних або ключів шифрування. Усі операції виконуються через відправку HTTP-запитів до сервісу.
2. Серверна частина (Service Provider):
    - Реалізована на платформі ASP.NET Core Web API.
    - Виступає постачальником сервісу, що надає бізнес-функції: авторизацію користувачів, валідацію транзакцій,

криптографічну обробку даних (шифрування/дешифрування AES) та збереження інформації у репозиторій.

- Забезпечує безпеку через перевірку JWT-токенів (Json Web Token) у заголовках запитів.

3. Протокол взаємодії (Communication & Data Exchange):
    - Обмін даними відбувається у форматі JSON.
    - Використовуються спеціальні об'єкти передачі даних DTO (Data Transfer Objects): TransactionDto (для запитів клієнта) та EncryptedResponse (для відповідей сервера). Це дозволяє відокремити внутрішню модель бази даних від зовнішнього контракту сервісу та забезпечити передачу лише необхідної інформації.
    - Конфіденційні дані (опис транзакцій) передаються та зберігаються у зашифрованому вигляді, розшифровка відбувається лише на стороні сервісу за запитом авторизованого клієнта.

Фрагменти програмного коду:

TransactionDTO.java

```
package ia32.koliada.finance.soa.dto;

import java.math.BigDecimal;

public class TransactionDTO {
   public BigDecimal amount;
   public String description;
   public String category;

   public TransactionDTO() {}
   public TransactionDTO(BigDecimal amount, String description, String
category) {
       this.amount = amount;
```

```
        this.description = description;
        this.category = category;
    }
}
```

SecurityService.java

```
@Service
public class SecurityService {
    private static final String KEY = "LabWorkSecretKey";
      public String generateToken(String user) {
        return "TOKEN-" + UUID.randomUUID();
    }

      public String encrypt(String data) {
        try {
            SecretKeySpec key = new SecretKeySpec(KEY.getBytes(), "AES");
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.ENCRYPT_MODE, key);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes()));
        } catch (Exception e) { throw new RuntimeException(e); }
    }
   }
```

TransactionController.java

```
@RestController
@RequestMapping("/api/transactions")
public class TransactionController {
    private final Repository repo;
    private final SecurityService security;

     public TransactionController(Repository repo, SecurityService security) {
        this.repo = repo;
        this.security = security;
    }

    @PostMapping
```

```
    public String create(@RequestBody TransactionDTO dto) {
            dto.description = security.encrypt(dto.description);
        repo.save(dto);
        return "Saved Successfully";
    }
}
```

FinanceClient.java

```
public void sendTransaction(TransactionDTO dto) {
    try {
        String jsonBody = mapper.writeValueAsString(dto);
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create("http://localhost:8080/api/transactions"))
                .header("Content-Type", "application/json")
                .header("Authorization", "Bearer " + this.token)
                .POST(HttpRequest.BodyPublishers.ofString(jsonBody))
                .build();

            HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
        System.out.println("Status: " + response.statusCode());
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Вихідний код:

https://github.com/mariiakoliada/TRPZ/tree/master/src/ia32/koliada/finance/soa

Питання до лабораторної роботи

1. Що таке клієнт-серверна архітектура?

Це архітектурна модель, в якій система розподіляється на дві частини: постачальників ресурсів або послуг (сервери) та споживачів (клієнтів).

Клієнти ініціюють сеанси зв'язку, надсилаючи запити, а сервери очікують на ці запити та обробляють їх, надсилаючи відповіді.

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA) – це модульний підхід до розробки програмного забезпечення, заснований на використанні розподілених, слабо пов'язаних (loose coupling) сервісів, оснащених стандартизованими інтерфейсами. Вона виникла як альтернатива монолітній архітектурі, де вся система розгортається як одне ціле.

3. Якими принципами керується SOA?

Сервіси мінімально залежать один від одного.Використання стандартизованих інтерфейсів та протоколів для взаємодії. Сервіси можуть бути реалізовані як обгортки навколо застарілих систем або як нові веб-служби.

4. Як між собою взаємодіють сервіси в SOA?

Сервіси взаємодіють між собою виключно за рахунок обміну повідомленнями (зазвичай по HTTP з використанням SOAP або REST). Важливою особливістю є те, що сервіси не мають спільних прямих інтеграцій для доступу до однієї інформації (наприклад, до однієї бази даних) кожен сервіс володіє своїми даними.

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

Згідно з концепцією SOA, сервіси реєструються на спеціальних сервісах (реєстрах), де будь-яка команда розробників може їх знайти. Для обміну даними часто використовують централізований компонент – шину даних (Enterprise Service Bus).

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

Переваги: Централізоване керування даними, легше забезпечення безпеки, простіше обслуговування (оновлення лише на сервері), розвантаження клієнтських машин.

Недоліки: Сервер може стати "вузьким місцем" (bottleneck) при великому навантаженні, єдина точка відмови (якщо сервер впаде – система не працює), висока вартість апаратного забезпечення сервера.

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

Переваги: Висока відмовостійкість (немає центрального сервера), легка масштабованість (кожен новий вузол додає ресурси), розподіл навантаження.

Недоліки: Складність керування та адміністрування, проблеми з безпекою та довірою до вузлів, непередбачувана доступність даних (вузол може вимкнутися).

8. Що таке мікро-сервісна архітектура?

Мікросервісна архітектура є подальшим розвитком сервіс-орієнтованої архітектури з використанням нових напрацювань в інформаційних технологіях. Це підхід, при якому додаток будується як набір невеликих, незалежних сервісів, кожен з яких виконує одну бізнес-функцію.

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

HTTP/HTTPS – для синхронної взаємодії.

gRPC – для високоефективної взаємодії між сервісами.

AMQP – для асинхронного обміну повідомленнями через черги.

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, це не є SOA. Це приклад багатошарової (Layered) архітектури монолітного додатку. У цьому випадку "сервіси" – це просто класи всередині однієї програми, які компілюються та розгортаються разом. SOA передбачає, що сервіси є розподіленими (можуть працювати на різних машинах) і взаємодіють через мережу, а не через виклик методів у пам'яті.

Висновок: під час виконання лабораторної роботи, я ознайомилася з видами взаємодії додатків (Client-Server, Peer-to-Peer, Serviceoriented Architecture), та реалізувала в проєктованій системі одну із архітектур Serviceoriented Architecture.