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: IRC-Клієнт

Виконала:
студентка групи ІА-32
Красоха В.О.

Перевірив:
Мягкий М.Ю.

Київ 2025

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.

Хід роботи

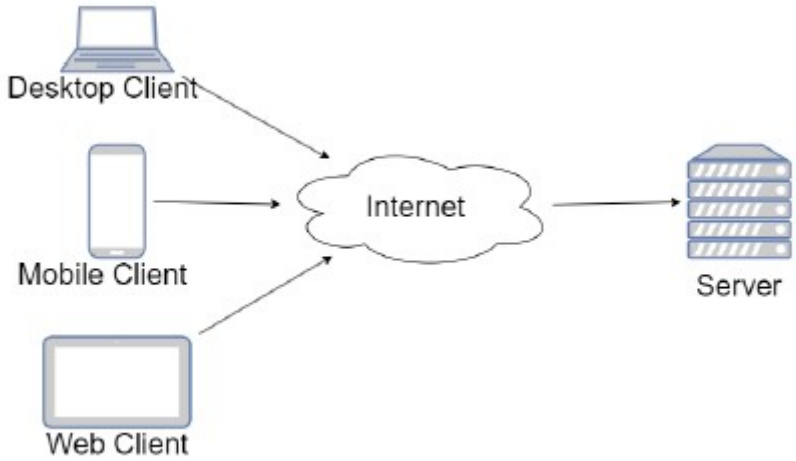
Клієнт-серверна архітектура

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.

Peer-to-Peer архітектура

Peer-to-Peer (P2P) архітектура – це модель мережевої взаємодії, в якій кожен вузол (комп'ютер або пристрій) є одночасно клієнтом і сервером. У цій архітектурі всі вузли мають рівні права та можливості для обміну даними, ресурсами або виконання завдань. На відміну від клієнт-серверної моделі, де є

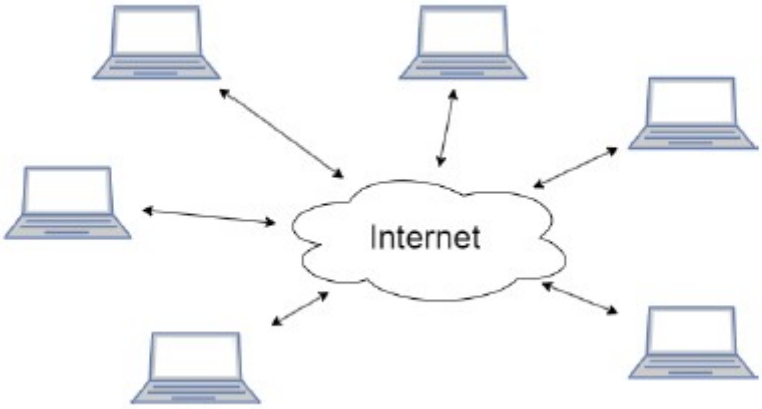
чітке розділення на клієнти й сервери, P2P-мережа дозволяє учасникам взаємодіяти безпосередньо, без необхідності в централізованому сервері.

Основними принципами P2P-архітектури є:

- Децентралізація – відсутність центрального сервера, що зменшує залежність від одного вузла, підвищуючи стійкість мережі до збоїв і атак.

- Рівноправність вузлів – кожен вузол може виконувати одночасно функції клієнта (отримувати ресурси) і сервера (надавати ресурси).

- Розподіл ресурсів – вузли надають доступ до своїх власних ресурсів, таких як обчислювальна потужність, дисковий простір або файли.



Основними сферами де peer-to-peer архітектура знайшла широке застосування є файлообмінники (BitTorrent), криптовалюти та інші блокчейн-технології, інтернет телефонія та відеоконференції (Skype, Zoom), розподілені обчислення (SETI@home, BOINC).

До основних проблемних зон можна віднести безпеку, синхронізацію даних та пошук ресурсів. Через централізацію складно контролювати дані, які передаються. Ефективність пошуку даних знижується зі збільшенням кількості вузлів у мережі і для підвищеня ефективності пошуку потрібно застосовувати спеціальні алгоритми.

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).

Мікросервісна архітектура є подальшим розвитком сервіс-орієнтованої архітектури з використанням нових напрацювань у інформаційних технологіях.

Мікро-сервісна архітектура.

Сама назва дає зрозуміти, що мікро-сервісна архітектура є підходом до створення серверного додатку як набору малих служб. Це означає, що

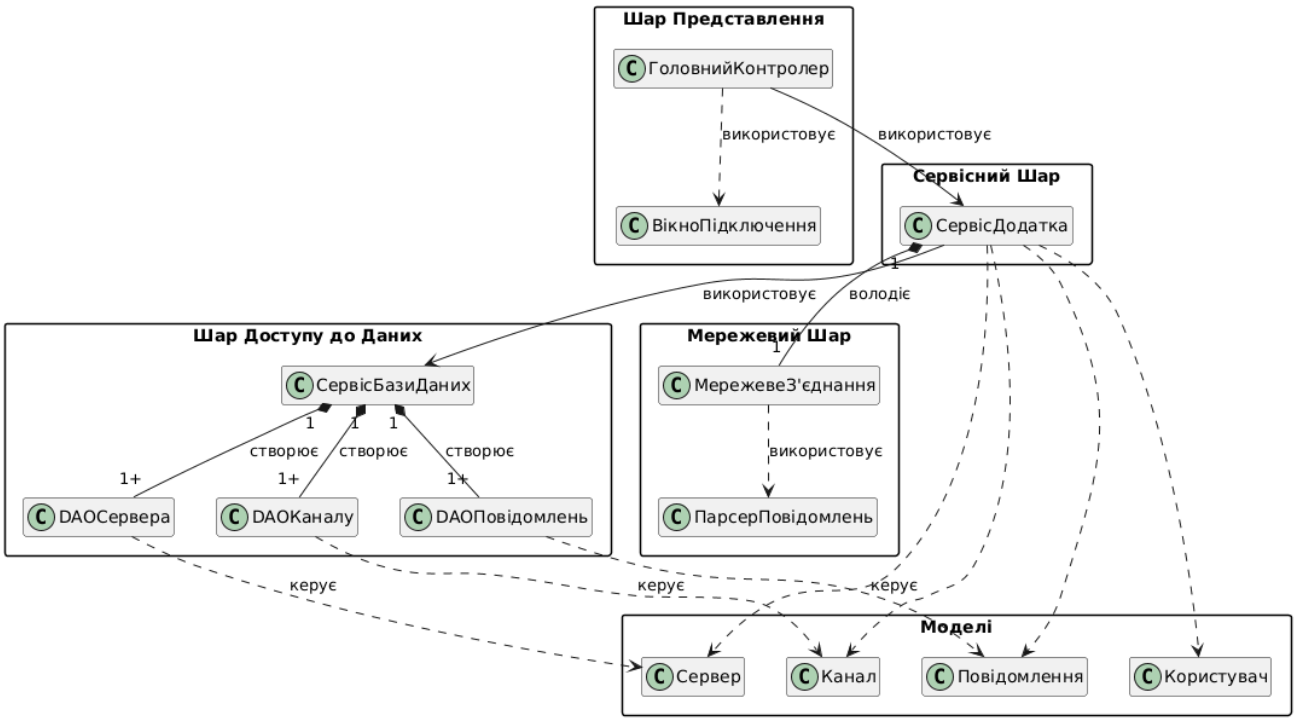
архітектура мікро-сервісів головним чином орієнтована на серверну частину, не дивлячись на те, що цей підхід так само використовується для зовнішнього інтерфейсу, де кожна служба виконується в своєму процесі і взаємодіє з іншими службами за такими протоколами, як HTTP/HTTPS, WebSockets чи AMQP. Кожен мікросервіс реалізує специфічні можливості в предметній області і свою бізнес-логіку в рамках конкретного обмеженого контексту, повинна розроблятись автономно і розвертатись незалежно.

Визначення мікросервісів із книги Ірaклі Надарейшвілі, Ронні Мітра, Метта Макларті та Майка Амундсена (О'Рейлі) «Архітектура мікросервісів»:

«Мікросервіс – це компонент із чітко визначеними межами, який можна розгортати незалежно, і підтримує взаємодію за допомогою зв'язку на основі повідомлень. Архітектура мікросервісів – це стиль розробки високоавтоматизованих систем програмного забезпечення, що легко розвивати та яке складається з мікросервісів, орієнтованих на певні можливості».

Мікросервіси забезпечують чудові можливості супроводження в величезних комплексних системах з високою масштабуємістю за рахунок створення додатків, заснованих на множині незалежно розгортуючих служб з автономними життєвими циклами.

Діаграма класів:



Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури.

Шар доступу до даних (DAO та Singleton)

Цей фрагмент демонструє використання патерна Singleton для управління підключенням та DAO для ізоляції SQL-логіки.

Файл DatabaseService.java (Singleton):

```
public class DatabaseService {
    private static DatabaseService instance;
    private Connection connection;

    private DatabaseService() {
        try {
            // Підключення до SQLite
            connection = DriverManager.getConnection("jdbc:sqlite:ircclient.db");
            initializeDatabase();
        } catch (SQLException e) { e.printStackTrace(); }
    }
```

```
    public static synchronized DatabaseService getInstance() {
        if (instance == null) instance = new DatabaseService();
        return instance;
    }
}
```

Файл MessageDao.java (Repository Pattern):

```
public void insert(Message message) throws SQLException {
     String sql = "INSERT INTO messages (channel_id, sender, content, timestamp) VALUES (?, ?, ?, ?)";
    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, message.getChannelId());
        pstmt.setString(2, message.getSender());
        pstmt.setString(3, message.getContent());
        pstmt.setTimestamp(4, Timestamp.valueOf(message.getTimestamp()));
        pstmt.executeUpdate();
    }
}
```

Сервісний шар (Бізнес-логіка)

Клас IRCService є центральним вузлом архітектури, який координує мережу та БД.

Файл IRCService.java:

```
public void sendMessage(String channel, String text) {
    // 1. Відправка через сокет (Мережевий шар)
    networkConnection.sendPrivmsg(channel, text);

    // 2. Збереження в базу (Шар даних)
    Message msg = new Message(channel, currentUser, text, LocalDateTime.now());
    messageDao.insert(msg);
```

```
    // 3. Оновлення UI через контролер
    mainController.displayMessage(msg);
}
```

Мережевий шар (TCP/IP Комунікація)

Фрагмент, що ілюструє фізичну комунікацію із зовнішнім сервером.

Файл IRCConnection.java:

```
public void connect(String host, int port) throws IOException {
    this.socket = new Socket(host, port);
                                this.writer = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
                                this.reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

    // Запуск потоку прослуховування сервера
    new Thread(this::listen).start();
}
```

Шар представлення (Controller)

Демонструє обробку подій користувача згідно з діаграмою послідовності.

Файл MainController.java:

```
@FXML
private void handleSendAction() {
    String text = messageInput.getText();
    if (text.startsWith("/")) {
        ircService.executeCommand(text); // Обробити як команду
    } else {
        ircService.sendMessage(currentChannel, text); // Обробити як повідомлення
    }
```

```
    messageInput.clear();
}
```

Доданий код:

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.ircclient</groupId>
    <artifactId>irc-client</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <javafx.version>17.0.2</javafx.version>
        <h2.version>2.2.224</h2.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>${javafx.version}</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>${javafx.version}</version>
        </dependency>

        <dependency>
            <groupId>org.xerial</groupId>
            <artifactId>sqlite-jdbc</artifactId>
            <version>3.46.0.0</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.11.0</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.8</version>
```

```
            <configuration>
                <mainClass>org.ircclient.Main</mainClass>
            </configuration>
        </plugin>
    </plugins>
</build>

</project>
```

styles.css:

```
/* Стилі для IRC клієнта */

.root {
    -fx-font-family: "Consolas", "Monaco", "Courier New", monospace;
    -fx-font-size: 12px;
}

/* Стилі для області чату */
.text-area {
    -fx-background-color: #1e1e1e;
    -fx-text-fill: #d4d4d4;
    -fx-font-family: "Consolas", "Monaco", "Courier New", monospace;
    -fx-font-size: 12px;
}

.text-area .content {
    -fx-background-color: #1e1e1e;
}

/* Стилі для списків */
.list-view {
    -fx-background-color: #252526;
    -fx-text-fill: #cccccc;
}

.list-cell {
    -fx-background-color: #252526;
    -fx-text-fill: #cccccc;
    -fx-padding: 5px;
}

.list-cell:selected {
    -fx-background-color: #094771;
}

.list-cell:hover {
    -fx-background-color: #2a2d2e;
}

/* Стилі для вкладок */
.tab-pane {
    -fx-background-color: #1e1e1e;
}

.tab {
    -fx-background-color: #2d2d30;
    -fx-text-fill: #cccccc;
}

.tab:selected {
```

```
    -fx-background-color: #1e1e1e;
}

.tab-label {
    -fx-text-fill: #cccccc;
}

/* Стилі для кнопок */
.button {
    -fx-background-color: #0e639c;
    -fx-text-fill: white;
    -fx-padding: 5px 15px;
}

.button:hover {
    -fx-background-color: #1177bb;
}

.button:pressed {
    -fx-background-color: #0a4d75;
}

/* Стилі для полів введення */
.text-field {
    -fx-background-color: #3c3c3c;
    -fx-text-fill: #cccccc;
    -fx-border-color: #3c3c3c;
    -fx-padding: 5px;
}

.text-field:focused {
    -fx-border-color: #0e639c;
}

/* Стилі для міток */
.label {
    -fx-text-fill: #cccccc;
}

/* Стилі для меню */
.menu-bar {
    -fx-background-color: #2d2d30;
}

.menu {
    -fx-text-fill: #cccccc;
}

.menu-item {
    -fx-background-color: #2d2d30;
    -fx-text-fill: #cccccc;
}

.menu-item:hover {
    -fx-background-color: #094771;
}

/* Стилі для панелей */
.vbox, .hbox {
    -fx-background-color: #1e1e1e;
}

.split-pane {
```

```
    -fx-background-color: #1e1e1e;
}

.split-pane-divider {
    -fx-background-color: #3c3c3c;
}

/* Стилі для ScrollPane */
.scroll-pane {
    -fx-background-color: #1e1e1e;
}

.scroll-bar {
    -fx-background-color: #1e1e1e;
}

.scroll-bar .thumb {
    -fx-background-color: #424242;
}

.scroll-bar .thumb:hover {
    -fx-background-color: #4e4e4e;
}
```

Контрольні питання:

1. Що таке клієнт-серверна архітектура?

Клієнт-серверна архітектура — це модель взаємодії, у якій клієнт ініціює запити, а сервер обробляє їх і надає відповіді, виконуючи зберігання даних, бізнес-логіку або обчислення.

2. Розкажіть про сервіс-орієнтовану архітектуру.

Сервіс-орієнтована архітектура (SOA) — це архітектурний підхід, у якому функціональність системи реалізується у вигляді незалежних сервісів, що взаємодіють між собою через стандартизовані інтерфейси.

3. Якими принципами керується SOA?

Основні принципи SOA:

- слабка зв’язаність сервісів
- повторне використання
- автономність сервісів
- стандартизовані інтерфейси
- незалежність від платформи

- можливість композиції сервісів

4. Як між собою взаємодіють сервіси в SOA?
Сервіси взаємодіють шляхом обміну повідомленнями через стандартизовані протоколи, зазвичай без прямої залежності від реалізації один одного.

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

- реєстри сервісів (наприклад, UDDI)
- документацію сервісів (WSDL, OpenAPI)
  Запити виконуються через визначені інтерфейси з використанням стандартних протоколів.

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

- централізоване управління даними
- простота адміністрування
- підвищена безпека

Недоліки:

- залежність від сервера
- обмежена масштабованість
- потенційна єдина точка відмови

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

- відсутність центрального вузла
- краща масштабованість
- підвищена стійкість до відмов

Недоліки:

- складність управління
- проблеми безпеки
- нерівномірний розподіл ресурсів

8. Що таке мікро-сервісна архітектура?
Мікросервісна архітектура — це підхід, у якому система складається з малих, автономних сервісів, кожен з яких реалізує окрему бізнес-функцію і може розгортатися незалежно.

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

- HTTP/HTTPS (REST)
- gRPC
- AMQP
- Kafka (подієва взаємодія)
- WebSocket

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 SOA, а багатошарова (layered) архітектура. У цьому випадку сервіси не є автономними, не мають власних контрактів і не взаємодіють через мережу, що є ключовими ознаками SOA.

Висновок: Під час виконання лабораторної роботи я вивчила види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувала в проєктованій системі одну із архітектур.