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.»

Виконав:

Студент групи ІА-32

Костінський Макарій

Перевірив:

Мягкий Михайло Юрійович

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

Клієнт-серверна архітектура (Client-Server): Модель взаємодії, де виділяються два типи компонентів: клієнти (ініціюють запити, забезпечують взаємодію з користувачем) та сервери (зберігають дані, виконують обчислення, надсилають відповіді). Розрізняють "товсті" клієнти (виконують частину логіки) та "тонкі" клієнти (передають все на сервер) .

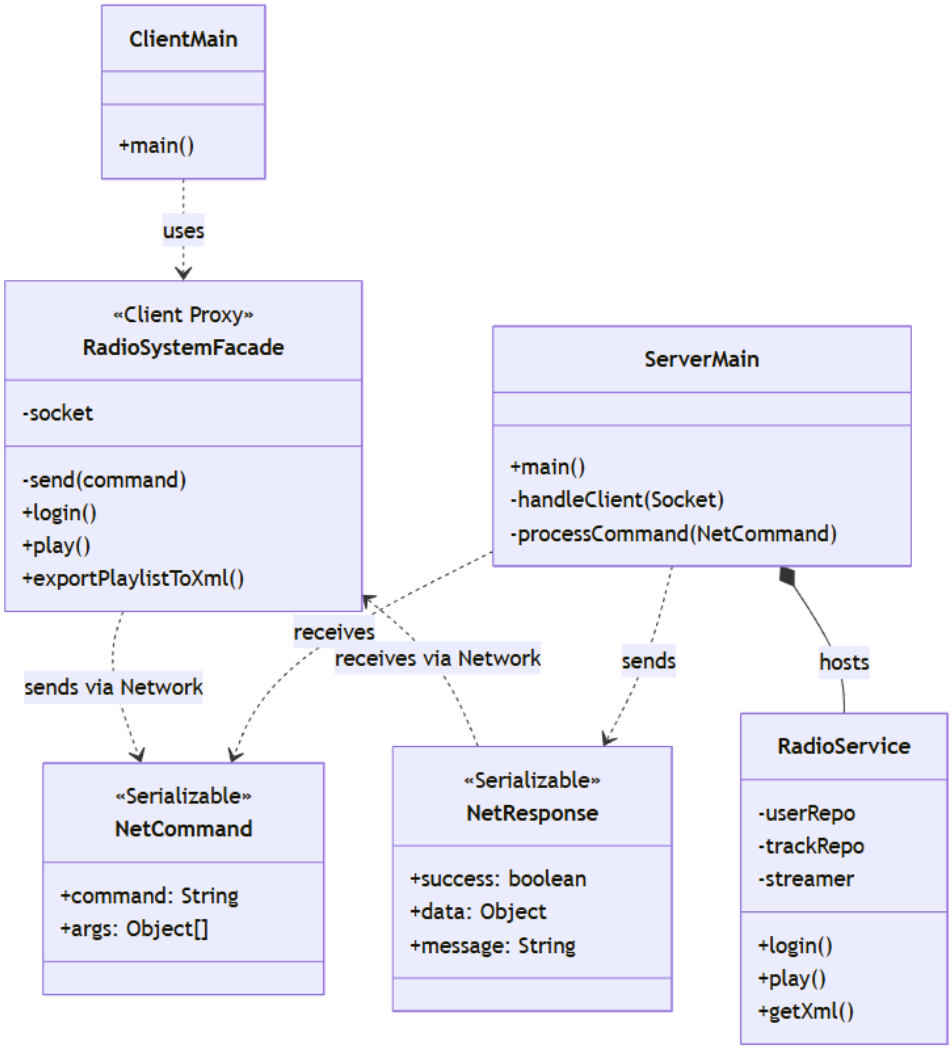
Peer-to-Peer архітектура (P2P): Децентралізована модель мережевої взаємодії, в якій кожен вузол є одночасно і клієнтом, і сервером. Всі вузли мають рівні права та можуть обмінюватися ресурсами безпосередньо, без участі центрального сервера .

Сервіс-орієнтована архітектура (SOA): Модульний підхід до розробки ПЗ, заснований на використанні розподілених, слабо пов'язаних сервісів, які взаємодіють через стандартизовані протоколи (наприклад, SOAP або REST). Сервіси надають певні бізнес-функції та можуть використовуватися різними додатками .

Мікросервісна архітектура (Microservices): Підхід до створення серверного додатку як набору невеликих, незалежних служб, кожна з яких виконується у своєму процесі та реалізує специфічну бізнес-логіку. Мікросервіси розгортаються незалежно та спілкуються через легковагі протоколи (HTTP, AMQP) .

Хід роботи

Діаграма класів Client-Server архітектури



Для забезпечення розподіленої роботи системи "Online Radio Station" було реалізовано клієнт-серверну архітектуру. Монолітний додаток було розділено на дві частини.

Серверна частина: Містить усю бізнес-логіку (RadioService), репозиторії даних та обробляє запити від клієнтів.

Клієнтська частина: Містить графічний інтерфейс користувача та спрощений фасад (RadioSystemFacade), який тепер виступає в ролі мережевого проксі, транслюючи дії користувача у команди серверу.

Обмін даними відбувається за допомогою серіалізованих об'єктів-команд (NetCommand) та відповідей (NetResponse).

Фрагменти коду по реалізації архітектури

Протокол обміну (NetCommand): Клас-обгортка для передачі команд через мережу.

```
package ua.kpi.iasa.onlineradio.net;
import java.io.Serializable;

public class NetCommand implements Serializable {
    private String command;
    private Object[] args;

    public NetCommand(String command, Object... args) {
        this.command = command;
        this.args = args;
    }
    public String getCommand() { return command; }
    public Object[] getArgs() { return args; }
}
```

Серверна частина (ServerMain): Слухає порт, приймає з'єднання та делегує виконання команд сервісу.

```
package ua.kpi.iasa.onlineradio.server;

import ua.kpi.iasa.onlineradio.models.IterationMode;
import ua.kpi.iasa.onlineradio.net.NetCommand;
import ua.kpi.iasa.onlineradio.net.NetResponse;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class ServerMain {
    private static final int PORT = 8888;
    private static RadioService service;

    public static void main(String[] args) {
        service = new RadioService();
        service.setPlaylist(1);

        System.out.println("SERVER: Started on port " + PORT);

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            while (true) {
                Socket clientSocket = serverSocket.accept();
                new Thread(() -> handleClient(clientSocket)).start();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void handleClient(Socket socket) {
        try (ObjectInputStream in = new
ObjectInputStream(socket.getInputStream());
             ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream())) {

            NetCommand cmd = (NetCommand) in.readObject();
            System.out.println("SERVER: Received command " + cmd.getCommand());
```

```
            NetResponse response = processCommand(cmd);
            out.writeObject(response);

        } catch (Exception e) {
            System.err.println("SERVER Error: " + e.getMessage());
        }
    }

    private static NetResponse processCommand(NetCommand cmd) {
        try {
            switch (cmd.getCommand()) {
                case "LOGIN":
                    boolean res = service.login((String)cmd.getArgs()[0],
(String)cmd.getArgs()[1]);
                    return new NetResponse(res, service.getCurrentUser(), res ?
"OK" : "Fail");
                case "PLAY":
                    service.play();
                    return new NetResponse(true, null, "Playing");
                case "NEXT":
                    service.nextTrack();
                    return new NetResponse(true, null, "Next");
                case "GET_CURRENT":
                    return new NetResponse(true, service.getCurrentTrack(),
"OK");
                case "LIKE":
                    service.likeCurrent();
                    return new NetResponse(true, null, "Liked");
                case "MODE":
                    service.changeMode((IterationMode) cmd.getArgs()[0]);
                    return new NetResponse(true, null, "Mode changed");
                case "REPORT":
                    return new NetResponse(true, service.getReport(1),
"Report");
                case "XML":
                    return new NetResponse(true, service.getXml(1), "XML");
                default:
                    return new NetResponse(false, null, "Unknown command");
            }
        } catch (Exception e) {
            return new NetResponse(false, null, e.getMessage());
        }
    }
}
```

Клієнтська частина (RadioSystemFacade): Тепер цей клас не містить логіки, а лише відправляє запити на сервер.

```
public class RadioSystemFacade {
    private static final String HOST = "localhost";
    private static final int PORT = 8888;
    private User currentUser;

    public RadioSystemFacade() {
    }

    private NetResponse send(String command, Object... args) {
        try (Socket socket = new Socket(HOST, PORT);
             ObjectOutputStream out = new
ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new
ObjectInputStream(socket.getInputStream())) {

            out.writeObject(new NetCommand(command, args));
```

```
            return (NetResponse) in.readObject();

        } catch (Exception e) {
            System.err.println("CLIENT Error: " + e.getMessage());
            return new NetResponse(false, null, "Connection Error");
        }
    }
…
    public void play() { send("PLAY"); }
…
}
```

Висновок:

У ході виконання лабораторної роботи засвоєно принципи побудови розподілених систем та вивчено особливості архітектур: «Client-Server», «Peer-to-Peer», «SOA» та «Microservices».

Для реалізації взаємодії компонентів у системі "Online Radio Station" було обрано клієнт-серверну архітектуру. Було здійснено розділення монолітного додатку на серверну частину (що містить бізнес-логіку та дані) та клієнтську частину (графічний інтерфейс). Реалізовано мережеву взаємодію між ними за допомогою TCP-сокетів та механізму серіалізації об'єктів, що дозволило керувати системою віддалено.