



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
**Технологія розроблення програмного
забезпечення**
Взаємодія компонентів системи

Виконав
студент групи ІА-31
Даценко Влад

Перевірив:
Мягкий Михайло
Юрійович

Київ 2025р.

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.

Завдання.

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.
4. 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, .NET Remoting на розсуд виконавця.

..25 Installer generator (iterator, builder, factory method, bridge, interpreter, client-server)

Генератор інсталяційних пакетів повинен мати якийсь спосіб налаштування файлів, що входять в установку, установки вікон з інтерактивними можливостями (галочка - створити ярлик на робочому столі; ввести в текстове поле деякі дані, наприклад, ліцензійний ключ і т.д.). Генератор повинен вивести один файл .exe або .msi.

Теоретичні відомості

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.

Хід роботи

В моїй лабораторній роботі реалізовано клієнтську і серверну частину додатків, а також загальну частину(middleware); зв'язок клієнтської і серверної частин за допомогою DAO.

Клієнт-серверна архітектура:



Я вирішив використати архітектуру client-server, щоб організувати взаємодію між різними частинами системи — клієнтом і сервером. Це дозволяє розділити відповідальність: клієнт відповідає за введення команд користувачем і відправку запитів, а сервер — за обробку цих запитів, роботу з базою даних і повернення результатів.

Клас Server створює серверний сокет, приймає підключення клієнтів і для кожного клієнта запускає окремий потік (ClientHandler), що забезпечує одночасну роботу з декількома клієнтами. Серверна частина використовує патерн DAO для ізоляції логіки доступу до бази даних, що спрощує підтримку та розширення системи.

Клас Client реалізує клієнтську частину: підключається до сервера через сокет, надсилає команди, отримує і виводить відповіді. Вся взаємодія між клієнтом і сервером відбувається через мережу, що дозволяє легко масштабувати систему, додавати нові клієнти або змінювати логіку обробки запитів на сервері без зміни клієнтського коду.

Завдяки такій структурі система стала гнучкою, розширюваною та зручною для підтримки, а також відповідає класичній моделі розподілених застосунків.

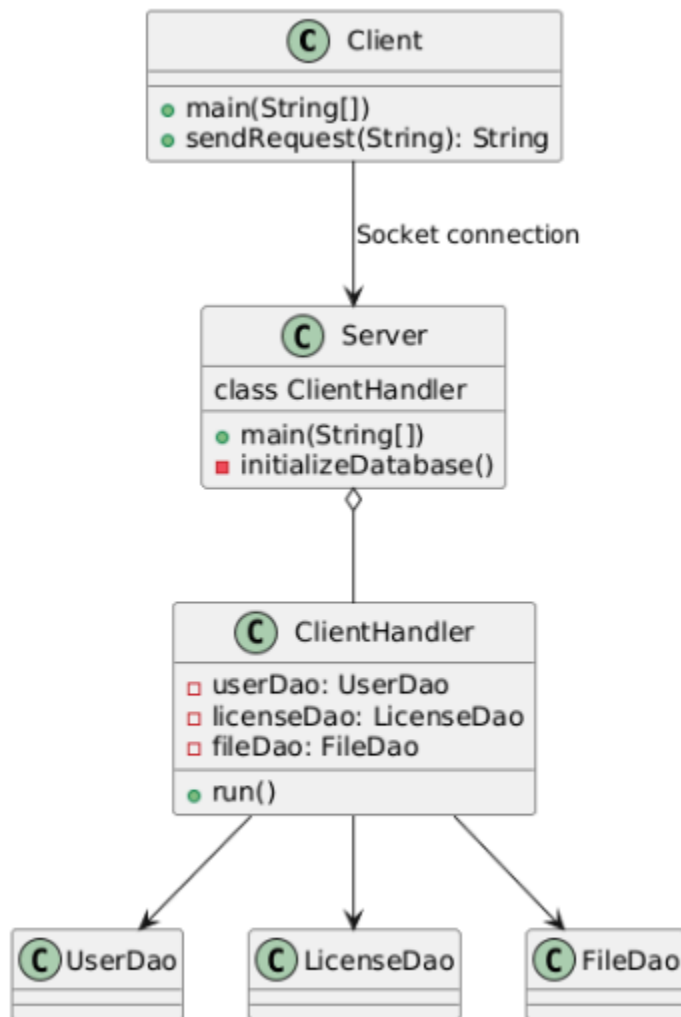


Рисунок 1. Структура клієнт-серверної архітектури з DAO

package org.example.server; You, 3 days ago • Initial commit ...

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.sql.Connection;
import java.sql.SQLException;

import org.example.server.dao.FileDao;
import org.example.server.dao.LicenseDao;
import org.example.server.dao.UserDao;
import org.example.utils.DatabaseUtil;
```

You, 3 days ago | You, 3 days ago | 1 author (You) | 1 author (You)

```
public class Server {
```

```
    private static final int PORT = 8081;
    private static Connection connection;
```

Run main | Debug main | Run | Debug

```
public static void main(String[] args) {
    try {
        initializeDatabase();

        ServerSocket serverSocket = new ServerSocket(PORT);
        System.out.println("Server started on port " + PORT);

        while (true) {
            Socket clientSocket = serverSocket.accept();
            System.out.println("New client connected: " + clientSocket.getInetAddress())
```

```

        new ClientHandler(clientSocket).start();
    }
} catch (IOException | SQLException e) {
    e.printStackTrace();
}
}

```

```

private static void initializeDatabase() throws IOException, SQLException {
    connection = DatabaseUtil.getInstance().getConnection();
    System.out.println(x: "Database connected.");
}

```

You, 3 days ago | You, 3 days ago | 1 author (You) | 1 author (You)

```

static class ClientHandler extends Thread {
    private Socket socket;
    private UserDao userDao;
    private LicenseDao licenseDao;
    private FileDao fileDao;

    public ClientHandler(Socket socket) throws IOException, SQLException {
        this.socket = socket;
        Connection connection = DatabaseUtil.getInstance().getConnection();
        this.userDao = new UserDao(connection);
        this.licenseDao = new LicenseDao(connection);
        this.fileDao = new FileDao(connection);
    }
}

```

```

@Override
public void run() {
    try (BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush: true)) {

        String request = in.readLine();
        System.out.println("Received request: " + request);

        String response = handleRequest(request);
        out.println(response);

    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

private String handleRequest(String request) {
    try {
        String[] parts = request.split(regex: " ");
        switch (parts[0]) {
            case "REGISTER":
                if (parts.length == 4) {
                    boolean license = Boolean.parseBoolean(parts[3]);
                    return userDao.registerUser(parts[1], parts[2], license);
                }
                break;
            case "LOGIN":
                if (parts.length == 3) {
                    return userDao.loginUser(parts[1], parts[2]);
                }
        }
    }
}

```

```

        return userDao.loginUser(parts[1], parts[2]);
    }
    break;
case "VALIDATE_LICENSE":
    if (parts.length == 2) {
        return licenseDao.isValidLicenseKey(parts[1]);
    }
    break;
case "SAVE_FILE":
    if (parts.length == 7) {
        return fileDao.saveFile(parts);
    }
    break;
default:
    return "Invalid request";
}
} catch (Exception e) {
    return "Error: " + e.getMessage();
}
return "Invalid request format";
}
}
}

```

Рисунок 2. Клас Server

```

public class Client {

    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 8081;

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        try (BufferedReader console = new BufferedReader(new InputStreamReader(System.in))) {
            Client client = new Client();
            while (true) {
                System.out.print(s: "Enter command: ");
                String command = console.readLine();
                if (command.equalsIgnoreCase(anotherString: "exit")) {
                    System.out.println(x: "Exiting...");
                    break;
                }

                String response = client.sendRequest(command);
                System.out.println("Server response: " + response);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String sendRequest(String request) {
        try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), autoFlush: true)) {

            out.println(request);
            return in.readLine();
        } catch (IOException e) {
            e.printStackTrace();
            return "Error: Unable to connect to server.";
        }
    }
}

```

Рисунок 3. Клас Client

```

public class FileDao {
    private final Connection connection;
    public FileDao(Connection connection) {
        this.connection = connection;
    }

    public String saveFile(String[] parts) {
        String query = "INSERT INTO files (user_id, input_file_path, input_file_type, output_f
        |   "VALUES (?, ?, ?, ?, ?)";
        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setInt(parameterIndex: 1, Integer.parseInt(parts[1]));
            stmt.setString(parameterIndex: 2, parts[2]);
            stmt.setString(parameterIndex: 3, parts[3]);
            stmt.setString(parameterIndex: 4, parts[4]);
            stmt.setString(parameterIndex: 5, parts[5]);
            stmt.executeUpdate();
            return "File saved successfully";
        } catch (SQLException e) {
            return "Error: " + e.getMessage();
        }
    }

    public void updateInstallationResult(int fileId, String status, String errorMessage) {
        String query = "UPDATE files SET status = ?, error_message = ? WHERE id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setString(parameterIndex: 1, status);
            stmt.setString(parameterIndex: 2, errorMessage);
            stmt.setInt(parameterIndex: 3, fileId);
            stmt.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

Рисунок 4. Клас FileDao

You, 3 days ago | 1 author (You)

```
public class LicenseDao {
    private final Connection connection;
    public LicenseDao(Connection connection) {
        this.connection = connection;
    }
    public String isValidLicenseKey(String licenseKey) {
        String query = "SELECT COUNT(*) FROM license_keys WHERE license_key = ?";
        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setString(parameterIndex: 1, licenseKey);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next() && rs.getInt(columnIndex: 1) > 0) {
                    return "VALID_LICENSE";
                } else {
                    return "INVALID_LICENSE";
                }
            }
        } catch (SQLException e) {
            return "Error: " + e.getMessage();
        }
    }
}
```

Рисунок 5. Клас LicenseDao

```

public class UserDao {
    private final Connection connection;
    public UserDao(Connection connection) {
        this.connection = connection;
    }
    public String registerUser(String username, String password, boolean license) {
        String query = "INSERT INTO users (username, password, license) VALUES (?, ?, ?)";
        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setString(parameterIndex: 1, username);
            stmt.setString(parameterIndex: 2, password);
            stmt.setBoolean(parameterIndex: 3, license);
            stmt.executeUpdate();
            return "User registered successfully";
        } catch (SQLException e) {
            return "Error: " + e.getMessage();
        }
    }

    public String loginUser(String username, String password) {
        try {
            String userExistsQuery = "SELECT id FROM users WHERE username = ?";
            try (PreparedStatement userExistsStmt = connection.prepareStatement(userExistsQuery)) {
                userExistsStmt.setString(parameterIndex: 1, username);
                try (ResultSet userExistsRs = userExistsStmt.executeQuery()) {
                    if (!userExistsRs.next()) {
                        return "Login failed: User does not exist.";
                    }
                }
            }
        }
    }
}

```

Рисунок 6. Клас UserDao

Висновок: У ході виконання лабораторної роботи було реалізовано архітектуру Client-Server, що дозволяє організувати взаємодію між клієнтською та серверною частинами системи через мережу. Основна ідея використання цієї архітектури полягає у чіткому розділенні відповідальності: клієнт відповідає за введення та надсилання команд, а сервер — за обробку запитів, роботу з базою даних і повернення результатів.

Завдяки такій структурі стало можливим легко масштабувати систему, додавати нові клієнти або змінювати логіку обробки запитів на сервері без зміни клієнтського коду. Використання шаблону DAO на сервері спростило роботу з базою даних і підвищило підтримуваність коду. Реалізація Client-Server у даній роботі є виправданою і продемонструвала свою ефективність для організації розподілених застосунків та взаємодії між різними компонентами системи.