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»

Виконала: Лапа Руслана Ігорівна

студент групи ІА-11

Дата здачі

Захищено з балом

Перевірив:

ст. вик. кафедри ІСТ

Колеснік В. М.

Київ 2023

Тема: Різні види взаємодії додатків: CLIENT-SERVER, PEER-TO-PEER, SERVICE-ORIENTED ARCHITECTURE.

Мета: Реалізувати функціонал для роботи в розподіленому оточенні (логіку роботи). Реалізувати взаємодію розподілених частин.

Хід роботи:

..8 Powershell terminal (strategy, command, factory method, template method, interpreter, client-server)

Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

1. Реалізувати функціонал для роботи в розподіленому оточенні (логіку роботи).

Рис. 1 – Вигляд client-server architecture

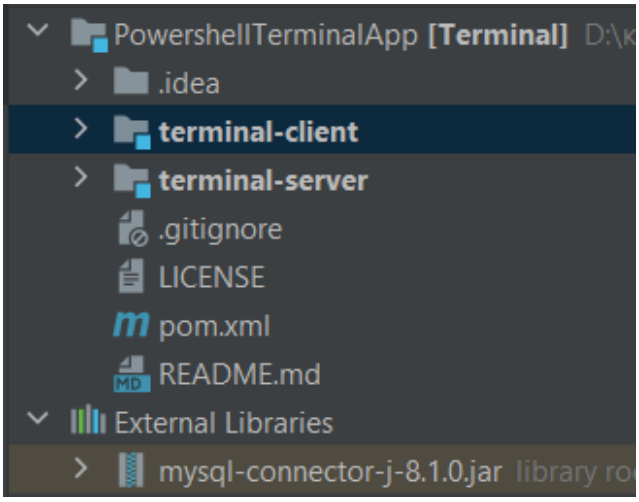

Рис. 1 – Структура client-server додатку

2. Реалізація client-server архітектури

ClientApp.java

```
package com.example.terminal;

import ...


public class ClientApp extends Application {

@Override
public void start(Stage primaryStage) throws Exception {

    FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "/sample.fxml"));
    Parent root = loader.load();

    RemoteController controller = loader.getController();
    controller.setServerUrl("http://localhost:55555");
    primaryStage.setOnCloseRequest(event -> controller.onCloseWindow());

    primaryStage.setTitle("Terminal App");
    primaryStage.setScene(new Scene(root,  v: 800,  v1: 600));
    primaryStage.show();
}

    public static void main(String[] args) { launch(args); }
}
```

ServerApp.java

```
public static void main(String[] args) {
    Javalin app = Javalin.create().start( port: 55555);
    app.delete( path: "/close-window", ServerApp ::closeWindowHandler);
    app.get( path: "/fetch-command-history", ServerApp::fetchCommandHistoryHandler);
    app.post( path: "/execute-command", ServerApp::executeHandler);

}
```

RemoteController.java

```java
private void executeDateFormatCommandLogic(String commandText, TextFlow textFlow) {

    try {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(serverUrl + "/execute-command"))
                .header( name: "Content-Type",  value: "text/plain")
                .POST(HttpRequest.BodyPublishers.ofString(commandText))
                .build();

        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

        String commandOutput = response.body();

        Syntax pathSyntax = new PathSyntax();
        pathSyntax.highlightSyntax(commandOutput, textFlow);

        scrollPane.setVvalue(1.0);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Висновок: у цій лабораторній роботі я ознайомилась із client-server архітектурою і реалізувала взаємодію розподілених частин.