|  |  |  |  |
| --- | --- | --- | --- |
| ***SPRAWOZDANIE NR 3*** | | | |
| Nazwa ćwiczenia | APLIKACJA TYPU CLIENT SERWER Z WYKORZYSTANIEM GNIAZD TCP/IP ORAZ WIELOWĄTKOWOŚCI W JAVIE | | Obraz zawierający tekst  Opis wygenerowany automatycznie |
| Przedmiot | Zaawansowane programowanie obiektowe | |
| Student  grupa | Paweł Jońca gr 7 122348 | |
| Data ćwiczeń | 01.06 | 01.06 | Data oddania sprawozdania |

Rozwiązanie zadania

Obraz zawierający tekst, oprogramowanie, Ikona komputerowa, System operacyjny

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

QuizzClient

import javax.swing.\*;  
import java.awt.\*;  
import java.io.\*;  
import java.net.\*;  
  
public class QuizClient extends JFrame {  
 private JTextField nicknameField;  
 private JTextField responseField;  
 private JButton submitButton;  
 private JTextArea messageArea;  
 private static final String SERVER\_HOST = "localhost";  
 private static final int SERVER\_PORT = 5000;  
  
 public QuizClient() {  
 super("Quiz Client");  
 initializeUI();  
 setVisible(true);  
 }  
  
 private void initializeUI() {  
 setSize(450, 350);  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 JPanel inputPanel = new JPanel(new GridBagLayout());  
 GridBagConstraints gbc = new GridBagConstraints();  
 gbc.insets = new Insets(5, 5, 5, 5);  
 gbc.fill = GridBagConstraints.HORIZONTAL;  
  
 JLabel nicknameLabel = new JLabel("Nickname:");  
 nicknameField = new JTextField(15);  
  
 JLabel responseLabel = new JLabel("Answer:");  
 responseField = new JTextField(15);  
  
 submitButton = new JButton("Submit");  
 submitButton.addActionListener(e -> sendResponse());  
  
 messageArea = new JTextArea();  
 messageArea.setEditable(false);  
  
 gbc.gridx = 0;  
 gbc.gridy = 0;  
 inputPanel.add(nicknameLabel, gbc);  
  
 gbc.gridx = 1;  
 inputPanel.add(nicknameField, gbc);  
  
 gbc.gridx = 0;  
 gbc.gridy = 1;  
 inputPanel.add(responseLabel, gbc);  
  
 gbc.gridx = 1;  
 inputPanel.add(responseField, gbc);  
  
 gbc.gridx = 0;  
 gbc.gridy = 2;  
 gbc.gridwidth = 2;  
 inputPanel.add(submitButton, gbc);  
  
 add(inputPanel, BorderLayout.NORTH);  
 add(new JScrollPane(messageArea), BorderLayout.CENTER);  
 }  
  
 private void sendResponse() {  
 String nickname = nicknameField.getText().trim();  
 String response = responseField.getText().trim();  
  
 if (nickname.isEmpty() || response.isEmpty()) {  
 displayMessage("Both nickname and answer are required!");  
 return;  
 }  
  
 try (Socket socket = new Socket(SERVER\_HOST, SERVER\_PORT);  
 PrintWriter writer = new PrintWriter(socket.getOutputStream(), true)) {  
 writer.println(nickname + "|" + response);  
 displayMessage("Sent: " + response);  
 responseField.setText("");  
 } catch (IOException e) {  
 displayMessage("Connection error: " + e.getMessage());  
 }  
 }  
  
 private void displayMessage(String message) {  
 messageArea.append(message + "\n");  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.invokeLater(QuizClient::new);  
 }  
}

QuizServer

import javax.swing.\*;  
import java.awt.\*;  
import java.io.\*;  
import java.net.\*;  
import java.util.\*;  
import java.util.List;  
import java.util.concurrent.\*;  
  
public class QuizServer extends JFrame {  
 private JTextArea logArea;  
 private ServerSocket serverSocket;  
 private Map<String, String> questions = new LinkedHashMap<>();  
 private int currentQuestionIndex = 0;  
 private boolean isGameOver = false;  
  
 private final BlockingQueue<String[]> answerQueue = new LinkedBlockingQueue<>();  
  
 public QuizServer() {  
 super("Quiz Server");  
 setupUI();  
 loadQuestionsFromFile();  
 startServerThreads();  
 setVisible(true);  
 }  
  
 private void setupUI() {  
 setSize(550, 400);  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 logArea = new JTextArea();  
 logArea.setEditable(false);  
 logArea.setFont(new Font("Courier New", Font.PLAIN, 14));  
  
 add(new JScrollPane(logArea), BorderLayout.CENTER);  
 }  
  
 private void loadQuestionsFromFile() {  
 try (BufferedReader reader = new BufferedReader(new FileReader("questions.txt"))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 String[] parts = line.split("\\|");  
 if (parts.length == 2) {  
 questions.put(parts[0].trim(), parts[1].trim());  
 }  
 }  
 logArea.append("Loaded " + questions.size() + " questions.\n");  
 } catch (IOException e) {  
 logArea.append("Error loading questions: " + e.getMessage() + "\n");  
 }  
 }  
  
 private void startServerThreads() {  
 new Thread(this::handleClientConnections).start();  
 new Thread(this::processAnswers).start();  
 }  
  
 private void handleClientConnections() {  
 try {  
 serverSocket = new ServerSocket(5000);  
 logArea.append("Server listening on port 5000...\n");  
  
 while (!isGameOver) {  
 Socket clientSocket = serverSocket.accept();  
 try (BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()))) {  
 String message = reader.readLine(); // Format: nickname|answer  
 String clientIP = clientSocket.getInetAddress().getHostAddress();  
  
 if (message != null && message.contains("|")) {  
 String[] data = message.split("\\|");  
 if (data.length == 2) {  
 answerQueue.put(new String[]{data[0], data[1], clientIP});  
 }  
 }  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 clientSocket.close();  
 }  
 } catch (IOException e) {  
 logArea.append("Server error: " + e.getMessage() + "\n");  
 }  
 }  
  
 private void processAnswers() {  
 List<String> questionList = new ArrayList<>(questions.keySet());  
  
 if (!questionList.isEmpty()) {  
 logArea.append("\nQUESTION 1: " + questionList.get(currentQuestionIndex) + "\n");  
 }  
  
 while (!isGameOver) {  
 try {  
 String[] answerData = answerQueue.take(); // Wait for an answer  
 String nickname = answerData[0];  
 String answer = answerData[1];  
 String clientIP = answerData[2];  
  
 String currentQuestion = questionList.get(currentQuestionIndex);  
 String correctAnswer = questions.get(currentQuestion);  
  
 if (answer.equalsIgnoreCase(correctAnswer)) {  
 logArea.append(nickname + " (" + clientIP + ") answered correctly!\n");  
 answerQueue.clear(); // Clear remaining answers  
 currentQuestionIndex++;  
  
 if (currentQuestionIndex < questionList.size()) {  
 logArea.append("\nQUESTION " + (currentQuestionIndex + 1) + ": " + questionList.get(currentQuestionIndex) + "\n");  
 } else {  
 logArea.append("\nGame over! All questions answered.\n");  
 isGameOver = true;  
 serverSocket.close();  
 }  
 } else {  
 logArea.append(nickname + " (" + clientIP + ") answered incorrectly.\n");  
 }  
 } catch (Exception e) {  
 logArea.append("Error processing answers: " + e.getMessage() + "\n");  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.invokeLater(QuizServer::new);  
 }  
}

plik z pytaniami

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.

Wnioski:

Wykonanie zadania sprawiło mi dużo trudności szczególnie w kwestii obsługi wielowątkowości na serwerze oraz poprawnej komunikacji między klientem a serwerem. Wykonując zadanie zrozumiałem w jaki sposób serwer może jednocześnie nasłuchiwać nowych połączeń i przetwarzać odpowiedzi przesyłane przez klienta.