External Project Report on Computer Networking (CSE3034)

CLIENT SERVER BASED SYSTEM TO GENERATE A DICTIONARY

Submitted by

Priyanshu Rajak	Reg. No.: 2141018039
Yash Yadav	Reg. No.: 2141025002
Harsh Narayan	Reg. No.: 2141014032
Anantsree Mohanty	Reg. No.: 2141013198
Subhajit Mohanty	Reg. No.: 2141013197

B. Tech. BRANCH 5th Semester (Section J)

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)
SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR, ODISHA

Declaration

We, the undersigned students of B. Tech. of **(CSE)** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled "**(Voice over Wi-Fi (VoWi-Fi) system)**" submitted to **Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking (CSE 3034)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Priyanshu Rajak
2141018039

Yash Yadav
2141025002

Harsh Narayan
2141014032

Anantsree Mohanty
2141013198

Subhajit Mohanty
2141013197

DATE:

PLACE:

Abstract

This project focuses on the design and implementation of a client-server based system using Java programming to facilitate dictionary generation for specified keywords and options. The system is intended for secure communication between clients and a dedicated server, where a Crunch Admin handles the generation of dictionaries and the subsequent upload to client-specified directories.

Key Features:

Network Administrator:

The system incorporates a dedicated network administrator, i.e., Crunch Admin, responsible for generating dictionaries based on client requests.
The generated dictionary is securely uploaded to the directory specified by the client upon completion.

Understanding Bruteforce-Based Attacks:

The system is designed to comprehend and counter bruteforce-based attacks, ensuring enhanced security during the dictionary generation process.
Measures are implemented to detect and prevent potential security threats, providing a robust defense against unauthorized access.

User Interface (GUI):

A user-friendly graphical user interface (GUI) is developed to facilitate seamless interactions between clients and the server.
Clients can easily request files, while the server can create new files or send existing ones through intuitive controls within the GUI.

Table of contents :

Introduction

The client-server model is a computing architecture where tasks are divided between two types of entities: clients and servers. Clients
initiate requests for services, such as accessing files or running applications, and servers fulfill these requests by providing the requested resources or performing the necessary computations. This model enables efficient resource utilization, centralized management, and facilitates scalable and distributed computing environments. Clients and servers communicate over a network using standardized protocols, creating a structured and organized system for information exchange and service delivery.

Key Objectives:

1.**Socket Programming:** - Develop a robust client-server architecture using Java's socket programming capabilities. This enables
bidirectional communication between the server (Crunch Admin) and multiple clients.

2. **Graphical User Interface (GUI):** - Implement a user-friendly GUI using Java Swing on the client side. This GUI facilitates
seamless interactions with the server, allowing users to request dictionaries and perform other operations.

3.Network Administration-Introduce the role of the server as a network administrator responsible for generating dictionaries. Clients
can request dictionaries based on specific keywords and options, fostering an understanding of dynamic content generation in a
networked environment.

4.File Operations-Enable the server to upload generated dictionaries to directories specified by clients. This involves handling file
operations and further illustrates the practical aspects of networked applications.

Methodology

The implementation of client-based server system to generate a string involves a systematic methodology that encompasses various stages to ensure a successful deployment and optimal performance.

1. Define the Protocol: Decide on a communication protocol between the client and server. It could be a simple text-based protocol or a more structured one like JSON.

2. Create Server Application: Implement a server application that listens for incoming client connections using Java's Server Socket class.

3. Implement Server-Side Socket Logic: Accept incoming client connections and spawn a new thread to handle each client request. Receive client requests, parse them according to the defined protocol, and perform the corresponding actions on the dictionary.

4. Develop Client Application: Create a graphical user interface (GUI) using Java's Swing or JavaFX for the client. Include components such as text fields, buttons, and labels for user input and output. Implement actions to send requests to the server and display the results.

5. Implement Client-Side Socket Logic: Establish a socket connection to the server using Java's Socket class. Send requests to the server, wait for responses, and update the GUI accordingly.

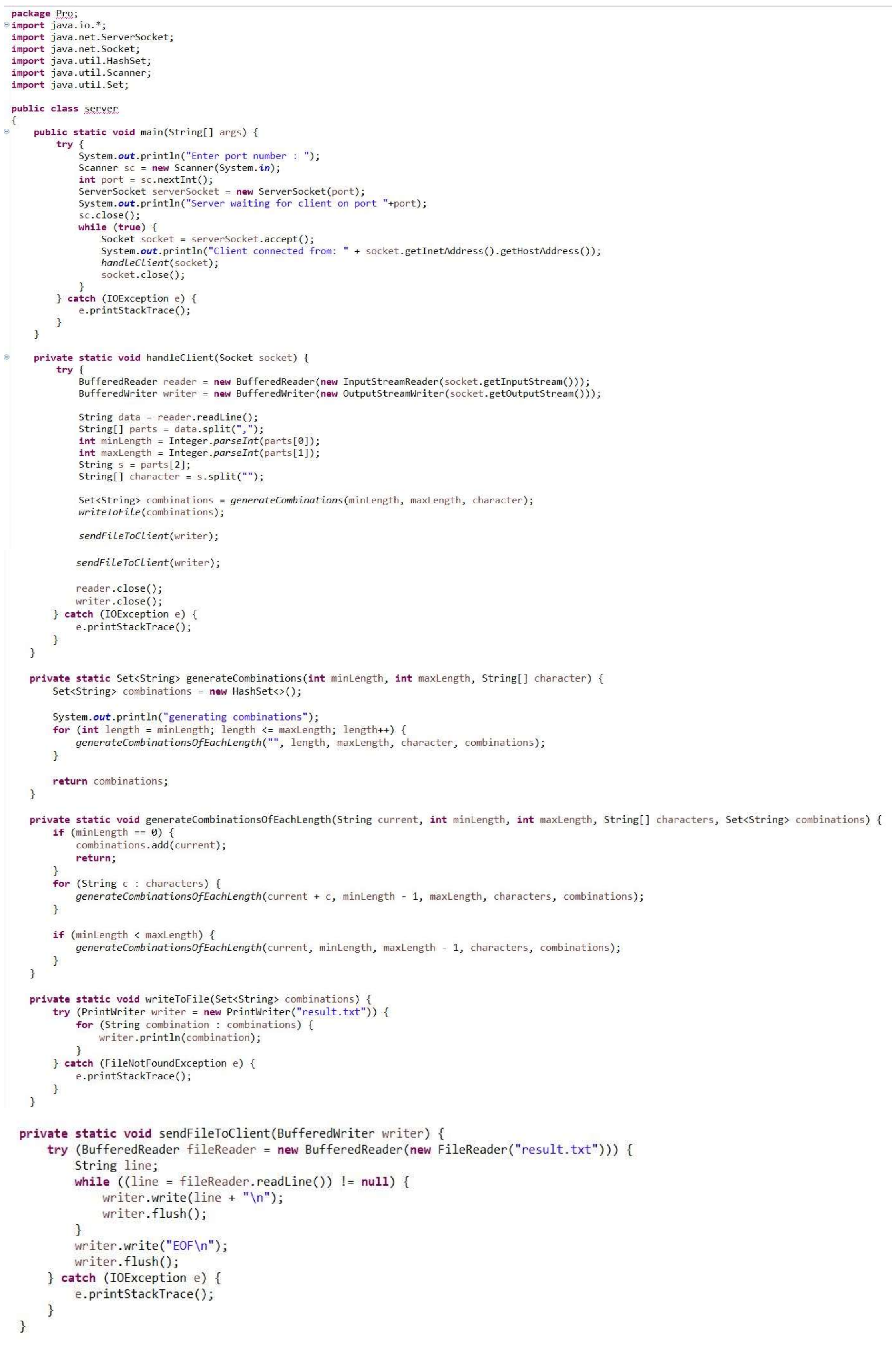
6. Define Communication Protocol: Clearly define the messages exchanged between the client and server.

7. Handle Errors and Exceptions: Implement error handling mechanisms on both the client and server sides.

Implementation

Below is the Java code for Server :

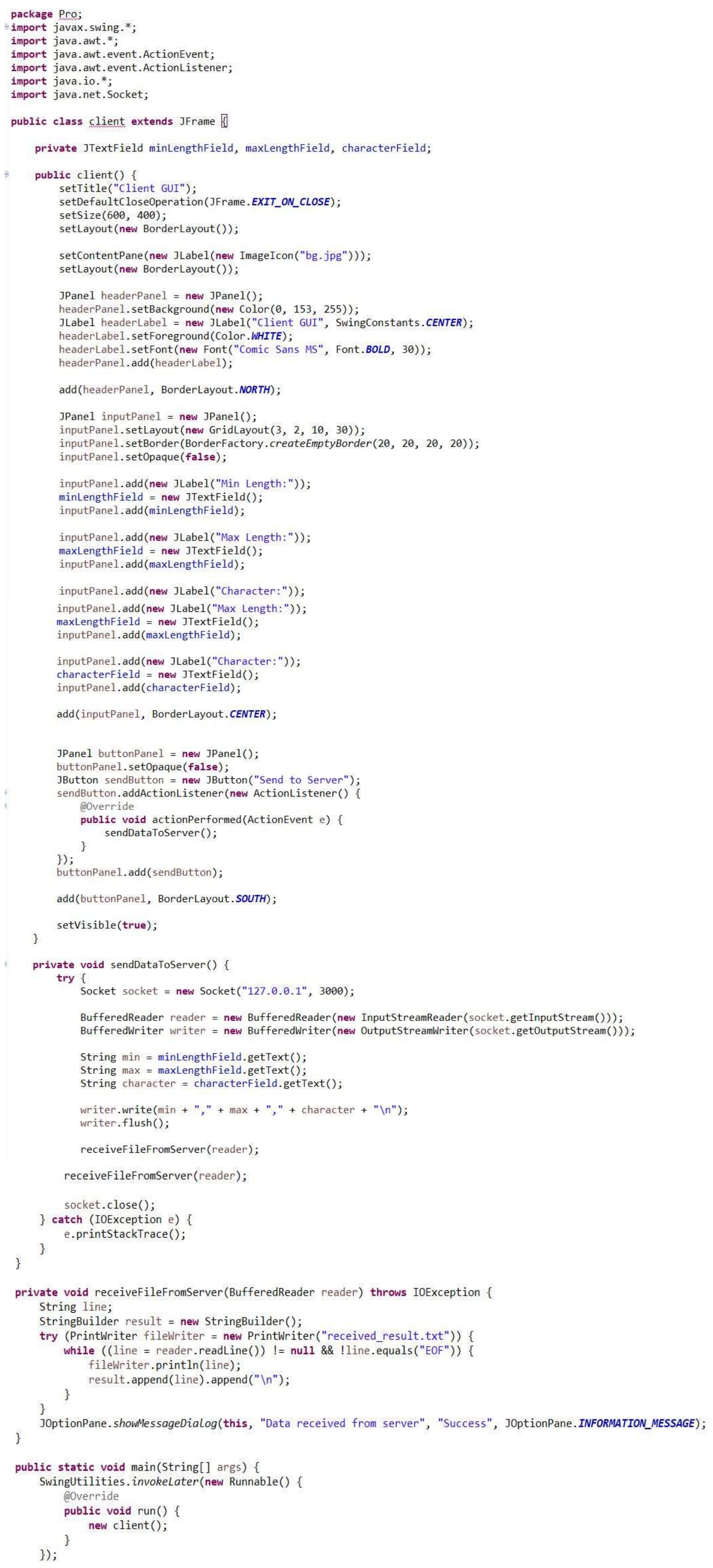
// Server Side



```java
package Pro;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class server
{
    public static void main(String[] args) {
        try {
            System.out.println("Enter port number : ");
            Scanner sc = new Scanner(System.in);
            int port = sc.nextInt();
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Server waiting for client on port "+port);
            sc.close();
            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected from: " + socket.getInetAddress().getHostAddress());
                handleClient(socket);
                socket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void handleClient(Socket socket) {
        try {
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

            String data = reader.readLine();
            String[] parts = data.split(",");
            int minLength = Integer.parseInt(parts[0]);
            int maxLength = Integer.parseInt(parts[1]);
            String s = parts[2];
            String[] character = s.split("");

            Set<String> combinations = generateCombinations(minLength, maxLength, character);
            writeToFile(combinations);

            sendFileToClient(writer);

            sendFileToClient(writer);

            reader.close();
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static Set<String> generateCombinations(int minLength, int maxLength, String[] character) {
        Set<String> combinations = new HashSet<>();

        System.out.println("generating combinations");
        for (int length = minLength; length <= maxLength; length++) {
            generateCombinationsOfEachLength("", length, maxLength, character, combinations);
        }

        return combinations;
    }

    private static void generateCombinationsOfEachLength(String current, int minLength, int maxLength, String[] characters, Set<String> combinations) {
        if (minLength == 0) {
            combinations.add(current);
            return;
        }
        for (String c : characters) {
            generateCombinationsOfEachLength(current + c, minLength - 1, maxLength, characters, combinations);
        }

        if (minLength < maxLength) {
            generateCombinationsOfEachLength(current, minLength, maxLength - 1, characters, combinations);
        }
    }

    private static void writeToFile(Set<String> combinations) {
        try (PrintWriter writer = new PrintWriter("result.txt")) {
            for (String combination : combinations) {
                writer.println(combination);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }

private static void sendFileToClient(BufferedWriter writer) {
    try (BufferedReader fileReader = new BufferedReader(new FileReader("result.txt"))) {
        String line;
        while ((line = fileReader.readLine()) != null) {
            writer.write(line + "\n");
            writer.flush();
        }
        writer.write("EOF\n");
        writer.flush();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Below is the Java code for Client :

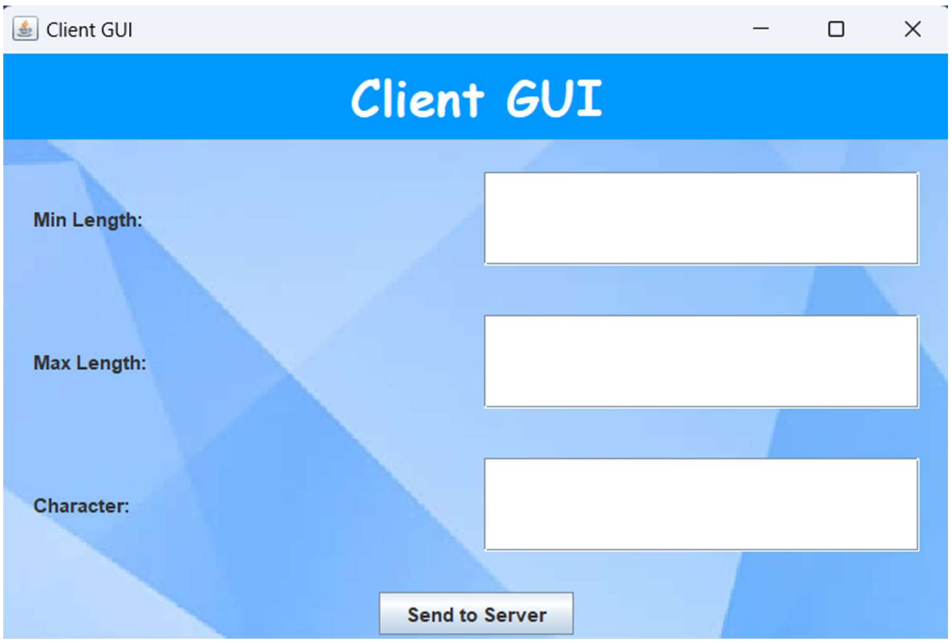
// Client Side



```java
package Pro;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.net.Socket;

public class client extends JFrame {

    private JTextField minLengthField, maxLengthField, characterField;

    public client() {
        setTitle("Client GUI");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 400);
        setLayout(new BorderLayout());

        setContentPane(new JLabel(new ImageIcon("bg.jpg")));
        setLayout(new BorderLayout());

        JPanel headerPanel = new JPanel();
        headerPanel.setBackground(new Color(0, 153, 255));
        JLabel headerLabel = new JLabel("Client GUI", SwingConstants.CENTER);
        headerLabel.setForeground(Color.WHITE);
        headerLabel.setFont(new Font("Comic Sans MS", Font.BOLD, 30));
        headerPanel.add(headerLabel);

        add(headerPanel, BorderLayout.NORTH);

        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new GridLayout(3, 2, 10, 30));
        inputPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
        inputPanel.setOpaque(false);

        inputPanel.add(new JLabel("Min Length:"));
        minLengthField = new JTextField();
        inputPanel.add(minLengthField);

        inputPanel.add(new JLabel("Max Length:"));
        maxLengthField = new JTextField();
        inputPanel.add(maxLengthField);

        inputPanel.add(new JLabel("Character:"));
        inputPanel.add(new JLabel("Max Length:"));
        maxLengthField = new JTextField();
        inputPanel.add(maxLengthField);

        inputPanel.add(new JLabel("Character:"));
        characterField = new JTextField();
        inputPanel.add(characterField);

        add(inputPanel, BorderLayout.CENTER);


        JPanel buttonPanel = new JPanel();
        buttonPanel.setOpaque(false);
        JButton sendButton = new JButton("Send to Server");
        sendButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                sendDataToServer();
            }
        });
        buttonPanel.add(sendButton);

        add(buttonPanel, BorderLayout.SOUTH);

        setVisible(true);
    }

    private void sendDataToServer() {
        try {
            Socket socket = new Socket("127.0.0.1", 3000);

            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

            String min = minLengthField.getText();
            String max = maxLengthField.getText();
            String character = characterField.getText();

            writer.write(min + "," + max + "," + character + "\n");
            writer.flush();

            receiveFileFromServer(reader);

        receiveFileFromServer(reader);

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

private void receiveFileFromServer(BufferedReader reader) throws IOException {
    String line;
    StringBuilder result = new StringBuilder();
    try (PrintWriter fileWriter = new PrintWriter("received_result.txt")) {
        while ((line = reader.readLine()) != null && !line.equals("EOF")) {
            fileWriter.println(line);
            result.append(line).append("\n");
        }
    }
    JOptionPane.showMessageDialog(this, "Data received from server", "Success", JOptionPane.INFORMATION_MESSAGE);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new client();
        }
    });
```

Results/Analysis

Client Page:

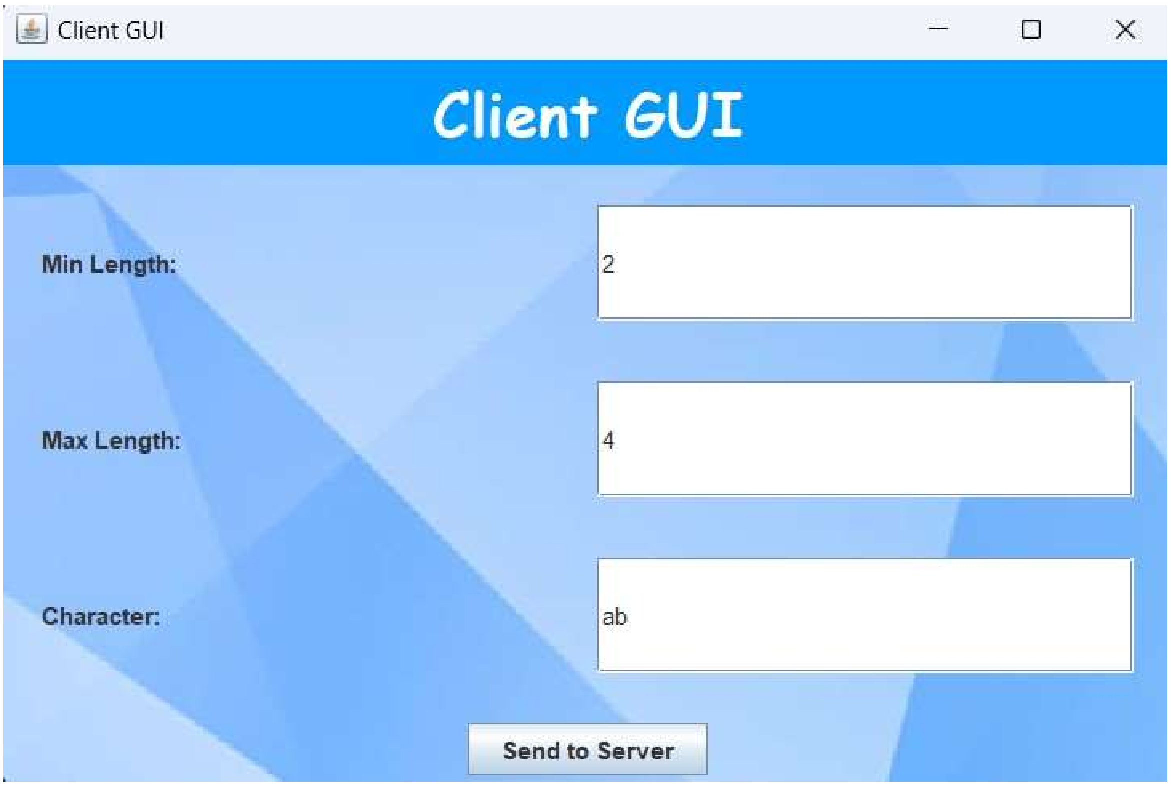


Min Length:

Max Length:

Character:

User Input:

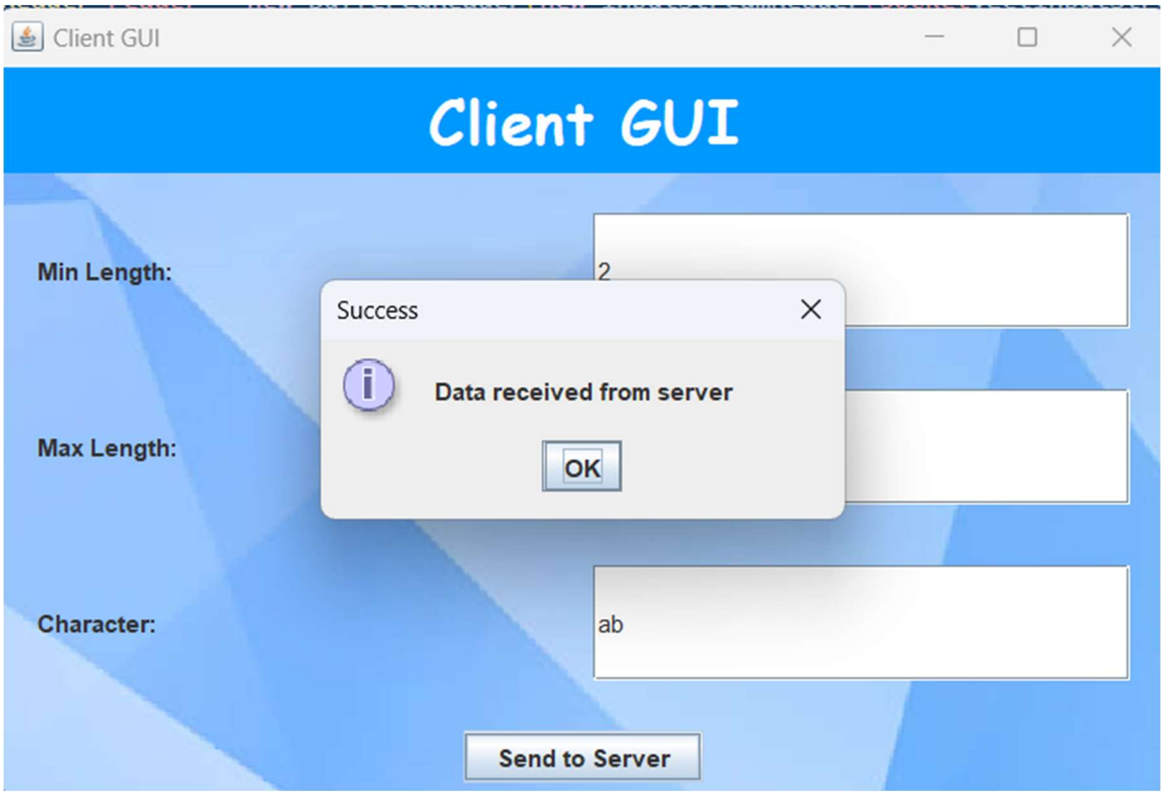


Min Length: 2

Max Length: 4

Character: ab

File Received:

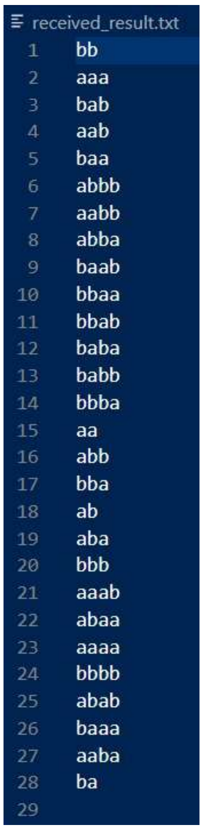


2

ab

Data received from server

Received File:



```
bb
aaa
bab
aab
baa
abbb
aabb
abba
baab
bbaa
bbab
baba
babb
bbba
aa
abb
bba
ab
aba
bbb
aaab
abaa
aaaa
bbbb
abab
baaa
aaba
ba

```

Conclusion

• The client-server model, with its clear separation of responsibilities between clients and servers, fosters efficient communication, promotes scalability, and enhances overall system reliability. This architectural approach has proven instrumental in building robust and responsive systems that cater to diverse user needs. In embracing the client-server model, organizations can achieve streamlined operations, centralized management, and a foundation for future growth, contributing to the development of powerful, dynamic, and user-friendly computing environments.

• In summary, this project serves as an educational tool for individuals seeking to understand and implement client-server systems, GUI development, and networked applications. It provides a foundation for further exploration into security aspects, scalability considerations, and enhanced user experiences. The combination of practical implementation and the introduction of security concepts makes this project a valuable resource for learning and expanding one's skills in the realm of networked systems and application development.

References

[1] Computer Networks, Andrew S. Tannenbaum, Pearson India.

[2] Java Network Programming by Harold, O’Reilly (Shroff Publishers).

[3] Java Complete Reference

[4] Youtube

[5] Wikipedia