**External Project Report on Computer Networking (CSE3034)**

**INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH**

**(FACULTY OF ENGINEERING)**

i

**SIKSHA ‘O’ ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR,**

**ODISHA**

Develop a client server-based system to generate a dictionary



**Submitted by**

|  |  |
| --- | --- |
| **Name 01** | **Reg. No.:** |
| **Name 02** | **Reg. No.:** |
| **Name 03** | **Reg. No.:** |
| **Name 04** | **Reg. No.:** |
| **Name 05** | **Reg. No.:** |

**B. Tech. CSIT 5th Semester (Section B )**

# Declaration

We, the undersigned students of B. Tech. of **Computer Science and Information Technology** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “**Develop a client-server based system where clients will ask Server (a Crunch Admin) to generate a dictionary for some given keywords and options**” 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.

|  |  |
| --- | --- |
| **(NAME1)**  **Registration No.:** | **(NAME2)**  **Registration No.:** |
| **(NAME3)**  **Registration No.:** | **(NAME4)**  **Registration No.:** |
| **(NAME5)**  **Registration No.:** | |

**DATE:**

**PLACE:**

# Abstract

This project revolves around the development of a client-server system in Java, emphasizing three key learning outcomes: GUI development, socket programming, and an understanding of brute-force attacks. Users interact with the system through a user-friendly Graphical User Interface (GUI), inputting keywords and options to request personalized dictionaries from the server, operated by the Crunch Admin.

The GUI serves as an intuitive platform for users, streamlining interactions and initiating requests for custom dictionary generation. The project delves into socket programming, establishing a robust communication framework between the Crunch Admin server and multiple clients. Proficiency in socket programming is crucial for building responsive and reliable networked applications.

In addition to core functionalities, the project introduces a Network Administrator role responsible for generating dictionaries and efficiently distributing them to specified client directories. This feature enhances system flexibility, allowing for centralized management of dictionary resources.

The project also explores brute-force attacks, shedding light on potential security threats and incorporating countermeasures to fortify the client-server system. This knowledge contributes to creating more secure applications, an essential aspect of today's cybersecurity landscape.

With a streamlined user interface facilitating operations and practical insights into socket programming and cybersecurity, this project offers a well-rounded learning experience in Java-based client-server systems. The combination of GUI design, socket programming, and security considerations makes it a valuable educational endeavor for those seeking hands-on experience in networked application development.

# Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **Serial**  **No.** | **Chapter No.** | **Title of the Chapter** | **Page No.** |
| 1. | 1 | Introduction |  |
| 2. | 2 | Problem Statement |  |
| 3. | 3 | Methodology |  |
| 4. | 4 | Implementation |  |
| 5. | 5 | Results and interpretation |  |
| 6. | 6 | Conclusion |  |
| 7. |  | References |  |
|  |  |  |  |

**1.Introduction**

This Java-based client-server system represents a significant advancement in network communication by employing Socket programming to enhance interactions between network administrators and clients.

The project places a strong emphasis on simplicity and functionality, featuring a user-friendly Graphical User Interface (GUI) developed using Java Swing. The GUI serves as an intuitive platform, enabling clients to seamlessly request files and execute various operations on the server, fostering a streamlined user experience.

At the system's core lies the Network Administrator module, empowering administrators to dynamically generate dictionaries tailored to specific keywords and options. These dictionaries are securely uploaded to client directories, contributing to heightened network security and refining password management processes.

Additionally, the project explores the nuances of bruteforce-based attacks, providing valuable insights into security mechanisms within networked systems.The overarching objective is to create a secure and efficient environment for collaboration in network management.

By prioritizing Java Swing for the GUI, the project aims to simplify the complexities of network administration, fostering improved communication and effectiveness in security operations.

This user-centric approach reflects a commitment to advancing networked systems through the integration of intuitive interfaces and robust security measures.

In essence, the project addresses the evolving needs of network administrators, offering a powerful tool for streamlined communication and enhanced security practices in contemporary network environments.

# 2. Problem Statement

I. Problem Explanation:

The project addresses the challenge of establishing a streamlined mechanism for user interaction through the console. The primary concern is the identification of elements or objects entered by the user, which triggers specific actions within the system.

Users, both network administrators and clients, need a simplified console interface to seamlessly request files, perform operations, and engage with the system. The desired outcome is the reflection of these interactions in the form of updated file content or database entries, providing a cohesive and user-friendly experience.

II. Constraints Highlight:

The project operates within certain constraints that warrant attention. These constraints include ensuring secure data transmission between the client and server, handling concurrent user requests effectively, and implementing a robust system for dictionary generation.

A significant constraint involves understanding and mitigating the risks associated with brute force-based attacks. This demands a comprehensive strategy for network security to safeguard against potential threats. Striking a balance between the need for simplicity in console interactions and the underlying operational complexity poses a critical design challenge.

The project recognizes these constraints as integral aspects of its development, and addressing them is essential for the successful and secure implementation of the envisioned console-based user interaction system.

# 3.Methodology

I. Algorithm/Pseudocode

function main():

port = readPortFromUser()

startServer(port)

function startServer(port):

serverSocket = createServerSocket(port)

print("Server waiting for client on port " + port)

while true:

socket = acceptConnection(serverSocket)

handleClient(socket)

closeConnection(socket)

function handleClient(socket):

data = readDataFromClient(socket)

minLength, maxLength, characters = parseData(data)

combinations = generateCombinations(minLength, maxLength, characters)

writeCombinationsToFile(combinations)

sendFileToClient(socket)

function generateCombinations(minLength, maxLength, characters):

// Recursive combination generation logic

function writeCombinationsToFile(combinations):

// Write combinations to "result.txt" file

function sendFileToClient(socket):

// Send contents of "result.txt" to client over the socket

# 4.Implementation

Program

1.Client:

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 ClientGUI extends JFrame {

private JTextField minLengthField, maxLengthField, characterField;

public ClientGUI() {

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:"));

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);

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 ClientGUI();

}

});

}

}

2. Server:

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();

try (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);

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();

}

}

}

# 5.Results & Interpretation

# Client Page:

# A screenshot of a computer Description automatically generated

# User Input:

# A screenshot of a computer Description automatically generated

# File Received:

# A screenshot of a computer Description automatically generated

# Received File:

# A blue screen with white text Description automatically generated

# 6.Conclusion

In conclusion, this project marks a significant achievement in enhancing user interactions through a GUI-based client-server system. Prioritizing simplicity and functionality, the system provides an intuitive and streamlined interface for both network administrators and clients to seamlessly engage with the application. Users can effortlessly request files, execute operations, and interact with the system, contributing to an enhanced user experience.

The project adeptly addresses constraints related to secure data transmission, concurrent user requests, and dictionary generation within the GUI framework. A robust approach to network security, including the mitigation of brute force-based attacks, underscores the project's commitment to safeguarding user data and system integrity.

As the project concludes, it not only achieves its primary goal of GUI-based interaction but also offers valuable insights into the complexities of network security within this graphical paradigm. The successful fusion of usability, functionality, and security in the GUI design positions this project as a foundational milestone for future advancements in client-server systems with graphical interfaces.

**References**

(as per the IEEE recommendations)

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

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

**[3] https://ieeexplore.ieee.org/abstract/document/8524573**

**[4] https://ieeexplore.ieee.org/abstract/document/6079253**