*Green University of Bangladesh*

*Department of Computer Science and Engineering (CSE)*
*Semester: (Spring, Year: 2024), B.Sc. in CSE (Day)*

# Toolkit for Encryption and Conversion Using Java Swing

*Course Title: Data Communication Lab*
*Course Code: CSE 308*
*Section: 222 D3*

Students Details

| Name | ID |
|---|---|
| Md.Naimul Islam | 221902056 |
| Sajal Bhuiyan | 221902064 |

*Submission Date: 08-06-2024*
*Course Teacher's Name: MR. Mahbubur Rahman*

[For teachers use only: Don't write anything inside this box]

| **Lab Project Report Status** | |
|---|---|
| **Marks:** | **Signature:** |
| **Comments:** | **Date:** |

# Contents

# Chapter 1

# Introduction

## 1.1 Overview

The "Toolkit for Encryption and Conversion Using Java Swing" project is a user-friendly desktop application built with Java Swing. It provides tools for encryption and data conversion tasks in a simple and intuitive interface. Users can perform operations like bit stuffing, character stuffing and IPv4 conversion easily without needing technical expertise. This toolkit aims to simplify data manipulation tasks and improve user productivity by offering a hassle-free experience within a desktop environment

## 1.2 Motivation

- Simplify encryption and conversion tasks.
- Make encryption techniques accessible to a wider audience.
- Provide a user-friendly interface for handling data encryption and conversion.
- Enable seamless implementation of bit stuffing, character stuffing and IPv4 conversion.
- Empower users to manage and secure their data efficiently and effectively.

## 1.3 Design Goals/Objectives

The design goals and objectives of the proposed project are as follows:

- The aim of this project is to implement what we learned in our Data Communication Lab.
- Develop an intuitive and user-friendly graphical interface.
- Implement robust encryption and conversion algorithms.

- Provide comprehensive documentation and user guides.
- Prioritize security and data integrity in all operations.

## 1.4 Problem Definition

he project addresses several challenges in data management and manipulation. These include the need for secure encryption methods to protect sensitive information, the necessity for efficient tools to convert data between various formats. By providing a unified solution to these issues, the project aims to simplify data-related tasks and enhance user experience.

### 1.4.1 Problem Statement

Our project seeks to tackle several challenges faced by users, including the complexities of encryption, the inconvenience of data conversion. By developing a user-friendly toolkit, we aim to streamline these processes and make them more accessible to users. This involves implementing robust encryption algorithms and simplifying data conversion tasks. The goal is to empower users to perform these tasks seamlessly, thereby enhancing their productivity and efficiency.

### 1.4.2 Challenges

The development of this system faces several challenges:

1. Making complicated encryption methods work smoothly.
2. Designing an interface that's easy for people to use.
3. Making sure data gets converted accurately.
4. Keeping everything secure from hackers.
5. Making the system run fast, even with a lot of data.
6. Handling mistakes and problems gracefully.

### 1.4.3 Complex Engineering Problem

The following table must be completed according to your above discussion in detail. The column on the right side should be filled only on the attributes you have chosen to be touched by your own project.

Table 1.1: Summary of the attributes touched by the mentioned projects

| **Name of the Attributes** | **Explain how to address** |
|---|---|
| **P1: Depth of knowledge required** | The project requires in-depth knowledge of network protocols, including bit stuffing, byte stuffing, IPv4 addressing, and error detection using Hamming code. Understanding these concepts is essential to implement and troubleshoot the functionalities accurately. |
| **P2: Range of conflicting requirements** | The project had to balance between accuracy and performance. For example, ensuring accurate error detection and correction while maintaining efficient processing speed posed a challenge. |
| **P3: Depth of analysis required** | Extensive analysis was conducted to ensure the correctness of algorithms, such as bit stuffing and Hamming code error detection. This included theoretical validation and practical testing to confirm the reliability of the implemented solutions. |
| **P4: Familiarity of issues** | Familiarity with network communication issues and error handling techniques was necessary to effectively implement and troubleshoot the various functionalities of the project. |
| **P5: Extent of applicable codes** | The project adhered to standard protocols and coding practices to ensure compatibility and reliability. This included following IPv4 standards and proper implementation of Hamming code algorithms. |
| **P6: Extent of stakeholder involvement and conflicting requirements** | Stakeholder involvement was minimal, primarily focusing on the project's technical requirements. Conflicting requirements were addressed through iterative development and regular feedback loops to ensure all needs were met. |
| **P7: Interdependence** | Different components of the project, such as bit stuffing and error detection, were interdependent. Ensuring seamless integration and proper functioning of each component required careful coordination and testing. |

## 1.5 Application

The "Toolkit for Encryption and Conversion Using Java Swing" project is designed to help users with tasks like securing their data, converting it between different formats. It provides easy-to-use tools for these tasks, making it accessible to users of all levels. The project aims to make data management simpler and more secure for everyone.

# Chapter 2

# Design/Development/Implementation of the Project

## 2.1 Introduction

This project focuses on designing, developing, and implementing various techniques for reliable data communication and error correction. These techniques include bit stuffing, character stuffing, IPv4 conversion, Parity Checker, and Hamming code error detection and correction. These methods are essential for maintaining data integrity and ensuring accurate data transmission in communication systems.

## 2.2 Project Details

In this section, we will elaborate on the specifics of each component of the project, detailing their functionality and significance.

### 2.2.1 Bit Stuffing and De-stuffing

Bit stuffing is used to prevent accidental occurrences of control information within the data stream. The process involves inserting a ’0’ bit after every sequence of five ’1’ bits to ensure that the data does not mimic control sequences. The de-stuffing process reverses this by removing the inserted ’0’ bits.

### 2.2.2 Character Stuffing and De-stuffing

Character stuffing involves adding special characters (ESC) before certain control characters within the data stream to distinguish between actual data and control information. This technique helps in maintaining the integrity of data by clearly differentiating it from control characters. The de-stuffing process involves removing these special characters to retrieve the original data.

### 2.2.3 IPv4 Conversion

IPv4 conversion translates a 32-bit binary IP address into a human-readable dotted decimal format. This process is crucial for network configuration and routing.

### 2.2.4 Parity checker Code

Parity checkers are used for error detection in data transmission. They add a parity bit to data, allowing the detection of single-bit errors. This method helps in ensuring the integrity of transmitted data

### 2.2.5 Hamming Code Error Detection and Correction

Using the parity bits in the Hamming code, this technique identifies and corrects errors in the received data. It ensures the integrity of the transmitted data by detecting single-bit errors and correcting them.

## 2.3 Implementation

In this section, we will delve into the detailed implementation of the project. We will discuss the implementation of each technique, including the algorithms, code structure, and tools used. The subsections will cover bit stuffing and de-stuffing, character stuffing and de-stuffing, IPv4 conversion, Hamming code generation, and Hamming code error detection and corrections.

#### The workflow

The workflow for implementing each technique involves designing algorithms, writing code, and testing the functionality. The project starts with bit and character stuffing, followed by IPv4 conversion and Hamming code generation. Error detection and correction using Hamming code is the final step.

#### Tools and libraries

The project utilizes various tools and libraries for development:

- **Java:** For implementing the algorithms and creating the user interface.
- **Swing:**For building the graphical user interface.
- **JUnit:** For testing the implemented code.

## 2.4 Algorithms

The algorithms and the programming codes in detail should be included .
Pseudo-codes are also encouraged very much to be included in this chapter for your project.

**Algorithm 1:** Bit Stuffing Algorithm

```
Input: Input Bit Stream
Output: Bit-stuffed Data Stream
Initialize count_of_ones to 0
for each bit in the input stream do
    if bit == 1 then
        Increment count_of_ones
        if count_of_ones == 5 then
            Append 0 to output stream
            Reset count_of_ones to 0
    else
        Reset count_of_ones to 0
    Append bit to output stream
Return bit-stuffed output stream
```

**Algorithm 2:** Bit De-stuffing Algorithm

```
Input: Bit-stuffed Data Stream
Output: Original Data Stream
Initialize count_of_ones to 0
for each bit in the stuffed stream do
    if bit == 1 then
        Increment count_of_ones
        Append bit to output stream
        if count_of_ones == 5 then
            Skip the next bit
            Reset count_of_ones to 0
    else
        Append bit to output stream
        Reset count_of_ones to 0
Return original output stream
```

**Algorithm 3:** Character Stuffing Algorithm

```
Input: Input Data Stream, FLAG, ESC
Output: Character-stuffed Data Stream
1 for each character in the input stream do
2     if character == FLAG or character == ESC then
3         Append ESC to output stream
4     Append character to output stream
5 Return character-stuffed output stream
```

**Algorithm 4:** Character De-stuffing Algorithm

```
Input: Character-stuffed Data Stream, ESC
Output: Original Data Stream
1 while not end of stream do
2     Read character
3     if character == ESC then
4         Read next character
5     Append character to output stream
6 Return original output stream
```

**Algorithm 5:** IPv4 Conversion Algorithm

```
Input: 32-bit Binary IP Address
Output: Dotted Decimal Notation
1 Initialize output string to empty
2 for each 8-bit segment in the binary IP address do
3     Convert 8-bit segment to decimal
4     Append decimal value to output with a dot separator
5 Remove the last dot from the output string
6 Return dotted decimal notation
```

**Algorithm 6:** Hamming Code Parity Checker Algorithm

```
Input: Data Bits
Output: Hamming Code with Parity Bits
1 Calculate the number of parity bits required
2 Initialize hamming code array with data and parity bits
3 for each parity bit position do
4     Set parity bit to 0 initially
5     for each bit covered by the parity bit do
6         Perform XOR operation to calculate parity
7     Set parity bit in the hamming code
8 Return hamming code
```

```
Algorithm 7: Hamming Code Error Detection and Correction Algorithm
    Input: Received Hamming Code
    Output: Corrected Hamming Code or Error Position
1   Initialize error position to 0
2   for each parity bit position do
3       Calculate parity value using the received hamming code
4       if parity value is incorrect then
5           Update error position by adding the parity bit position
6   if error position is 0 then
7       Return "No Error"
8   else
9       Invert the bit at error position
10      Return corrected hamming code
```

## Implementation details

### Home Page

```java
public class welcomePage extends JFrame implements
   ActionListener {


        JLabel headerLabel = new JLabel("ToolKit For
   Encryption and Conversion");

            new StuffingDeStuffing();
            frame.setVisible(false);
            }
        });
        frame.add(btnClass1);

        JButton btnClass2 = new JButton("IPv4 implementation"
   );

        btnClass2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

            new IPv4Implementation();
            frame.setVisible(false);
            }
        });
        frame.add(btnClass2);

        JButton btnClass3 = new JButton("Hamming Code");

        btnClass3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                 new HammingCode();
                frame.setVisible(false);
            }
        });
        frame.add(btnClass3);

        JButton btnClass4 = new JButton("Hamming Code Error
   Detection");
        ;
        btnClass4.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

            new HammingCodeErrorCorrection();
           frame.setVisible(false);
            }
        });
        frame.add(btnClass4);
     public void actionPerformed(ActionEvent e) {

    }

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

**Bit Stuffing and De-stuffing**

```
ng extends JFrame {

    public BitStuffingDeStuffing() {
        setTitle("Bit Stuffing and De-Stuffing");

        setDefaultCloseOperation(EXIT_ON_CLOSE);
        JLabel label = new JLabel("Bit Stuffing and De-
   Stuffing");
      );

        JButton stuffButton = new JButton("Stuff");
        JButton deStuffButton = new JButton("De-Stuff");
   );
        stuffButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String input = inputArea.getText();
                String output = bitStuffing(input);
                outputArea.setText(output);
            }
        });

        deStuffButton.addActionListener(new ActionListener()
```

```java
    {
            public void actionPerformed(ActionEvent e) {
                String input = inputArea.getText();
                String output = bitDeStuffing(input);
                outputArea.setText(output);
            }
        });
    }

    private String bitStuffing(String input) {
        StringBuilder stuffed = new StringBuilder();
        int count = 0;
        for (char bit : input.toCharArray()) {
            if (bit == '1') {
                count++;
                if (count == 5) {
                    stuffed.append('1');
                    stuffed.append('0');
                    count = 0;
                } else {
                    stuffed.append(bit);
                }
            } else {
                stuffed.append(bit);
                count = 0;
            }
        }
        return stuffed.toString();
    }

    private String bitDeStuffing(String input) {
        StringBuilder deStuffed = new StringBuilder();
        int count = 0;
        for (char bit : input.toCharArray()) {
            if (bit == '1') {
                count++;
                if (count == 5) {
                    count = 0;
                } else {
                    deStuffed.append(bit);
                }
            } else {
                deStuffed.append(bit);
                count = 0;
            }
        }
        return deStuffed.toString();
    }

    public static void main(String[] args) {
        new BitStuffingDeStuffing().setVisible(true);
    }
}
```

## Character Stuffing and De-stuffing

```

public class CharStuffingDeStuffing extends JFrame {

    public CharStuffingDeStuffing() {

        setTitle("Character Stuffing and De-Stuffing");
         JLabel label = new JLabel("Character Stuffing and De-Stuffing");
        JButton stuffButton = new JButton("Stuff");
        JButton deStuffButton = new JButton("De-Stuff");

        stuffButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String input = inputArea.getText();
                String output = charStuffing(input);
                outputArea.setText(output);
            }
        });

        deStuffButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String input = inputArea.getText();
                String output = charDeStuffing(input);
                outputArea.setText(output);
            }
        });
    }

    private String charStuffing(String input) {
        return input.replace("FLAG", "ESCFLAG").replace("ESC", "ESCESC");
    }

    private String charDeStuffing(String input) {
        return input.replace("ESCESC", "ESC").replace("ESCFLAG", "FLAG");
    }

    public static void main(String[] args) {
        new CharStuffingDeStuffing().setVisible(true);
    }
}
```

**IPv4 Menu**

```java

public class IPv4Implementation extends JFrame {

    public IPv4Implementation() {

        setTitle("IPv4 Implementation");
        getContentPane().setBackground(new Color(136, 196, 245));
        setSize(1000, 600);
        setLocationRelativeTo(null);
        setLayout(null);
        setVisible(true);
        setResizable(false);
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        JButton ipv4ConversionButton = new JButton("IPv4 Conversion");

        ipv4ConversionButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                new IPv4Converter();
                setVisible(false);
            }
        });
        add(ipv4ConversionButton);
        JButton ipv4DetailsButton = new JButton("IPv4 Details");

        ipv4DetailsButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                new IPv4Details();
                setVisible(false);
            }
        });
        add(ipv4DetailsButton);
    }

    public static void main(String[] args) {
        new IPv4Implementation();
    }
}
```

### IPv4 conversion

```java

public class IPv4Converter extends JFrame implements ActionListener {

   public IPv4Converter() {


      setTitle("IPv4 Implementation");


      Back.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {

            new IPv4Implementation();
            setVisible(false);
         }
      });
      add(Back);


     ImageIcon color_background = new ImageIcon("IMAGES/color.png");

      ipOutputLabel = new JLabel();

   public void actionPerformed(ActionEvent e) {
      String ipInput = ipInputField.getText();
      String selectedOption = (String) conversionOptions.getSelectedItem();

      if (e.getSource() == convertButton) {
         if ("Decimal to Binary".equals(selectedOption)) {
            convertDecimalToBinary(ipInput);
         } else if ("Binary to Decimal".equals(selectedOption)) {
            convertBinaryToDecimal(ipInput);
         }
      }
   }

   private void convertDecimalToBinary(String ipInput) {
      if (ipInput.matches("^\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}$")) {
         String[] octets = ipInput.split("\\.");
         StringBuilder binaryAddress = new StringBuilder();

         for (String octet : octets) {
            int decimal = Integer.parseInt(octet);
```

```java
                String binary = Integer.toBinaryString(decimal);
                binary = String.format("%8s", binary).replace(' ', '0');
                binaryAddress.append(binary).append('.');
            }

            ipOutputLabel.setText("Binary: " + binaryAddress.substring(0, binaryAddress.length() - 1));
        } else {
            ipOutputLabel.setText("Invalid IPv4 Decimal Address");
        }
    }

    private void convertBinaryToDecimal(String ipInput) {
        if (ipInput.matches("^(\\d{8}\\.){3}\\d{8}$")) {
            String[] octets = ipInput.split("\\.");
            StringBuilder decimalAddress = new StringBuilder();

            for (String octet : octets) {
                int decimal = Integer.parseInt(octet, 2);
                decimalAddress.append(decimal).append('.');
            }

            ipOutputLabel.setText("Decimal: " + decimalAddress.substring(0, decimalAddress.length() - 1));
        } else {
            ipOutputLabel.setText("Invalid IPv4 Binary Address");
        }
    }

    public static void main(String[] args) {
        new IPv4Converter();
    }
}
```

### Ipv4 Deatils

```java
package data_communiation;

public class IPv4Details extends JFrame implements ActionListener {


    public IPv4Details() {


```

```java

 public void actionPerformed(ActionEvent e) {
     String ipInput = ipInputField.getText();
     String selectedConversion = (String) conversionTypeComboBox.getSelectedItem();
     if (e.getSource() == calculateButton) {
         if ("Decimal to Binary Details".equals(selectedConversion)) {
             calculateBinaryDetails(ipInput);
         } else if ("Decimal to Decimal Details".equals(selectedConversion)) {
             calculateDecimalDetails(ipInput);
         }
     }
 }

 private void calculateBinaryDetails(String ipInput) {
     try {
         String[] parts = ipInput.split("/");
         if (parts.length != 2) {
             ipOutputLabel.setText("Invalid input format. Expected format: x.x.x.x/x");
             return;
         }

         String ipPart = parts[0];
         int prefixLength = Integer.parseInt(parts[1]);

         if (!ipPart.matches("^\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}$") || prefixLength < 0 || prefixLength > 32) {
             ipOutputLabel.setText("Invalid IPv4 Address or prefix length");
             return;
         }

         String[] octets = ipPart.split("\\.");
         int[] ipOctets = Arrays.stream(octets).mapToInt(Integer::parseInt).toArray();

         if (Arrays.stream(ipOctets).anyMatch(o -> o < 0 || o > 255)) {
             ipOutputLabel.setText("Invalid IPv4 Address");
             return;
         }

         StringBuilder binaryIP = new StringBuilder();
         for (int octet : ipOctets) {
             String binary = String.format("%8s", Integer.toBinaryString(octet)).replace(’ ’, ’0’);
             binaryIP.append(binary).append('.');
         }

         binaryIP.setLength(binaryIP.length() - 1);

         String binaryIPStr = binaryIP.toString();
         String subnetMask = generateSubnetMask(prefixLength);
         String networkAddress = calculateNetworkAddress(binaryIPStr, subnetMask);
         String broadcastAddress = calculateBroadcastAddress(networkAddress, subnetMask);
         char ipClass = determineClass(ipOctets[0]);
         int numberOfHosts = (int) Math.pow(2, 32 - prefixLength) - 2;

         String output = String.format(
                 "<html>IPv4: %s<br>Subnet Mask: %s<br>Network Address: %s<br> Broadcast Address: %s<br>Class: %s<br>Number of hosts: %d</html>",
                 binaryIPStr, subnetMask, networkAddress, broadcastAddress, ipClass, numberOfHosts
         );
         ipOutputLabel.setText(output);
     } catch (Exception ex) {
         ipOutputLabel.setText("An error occurred: " + ex.getMessage());
     }
 }

 private void calculateDecimalDetails(String ipInput) {
     try {
         String[] parts = ipInput.split("/");
         if (parts.length != 2) {
             ipOutputLabel.setText("Invalid input format. Expected format: x.x.x.x/x");
             return;
         }

         String ipPart = parts[0];
         int prefixLength = Integer.parseInt(parts[1]);

         if (!ipPart.matches("^\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}$") || prefixLength < 0 || prefixLength > 32) {
             ipOutputLabel.setText("Invalid IPv4 Address or prefix length");
             return;
         }

         String[] octets = ipPart.split("\\.");
```

```java
         int[] ipOctets = Arrays.stream(octets).mapToInt(Integer::parseInt).toArray();

         if (Arrays.stream(ipOctets).anyMatch(o -> o < 0 || o > 255)) {
             ipOutputLabel.setText("Invalid IPv4 Address");
             return;
         }

         String subnetMask = generateSubnetMaskDecimal(prefixLength);
         String networkAddress = calculateNetworkAddressDecimal(ipPart, subnetMask);
         String broadcastAddress = calculateBroadcastAddressDecimal(networkAddress, subnetMask);
         char ipClass = determineClass(ipOctets[0]);
         int numberOfHosts = (int) Math.pow(2, 32 - prefixLength) - 2;

         String output = String.format(
                 "<html>IPv4: %s<br>Subnet Mask: %s<br>Network Address: %s<br> Broadcast Address: %s<br>Class: %s<br>Number of hosts: %d</html>",
                 ipPart, subnetMask, networkAddress, broadcastAddress, ipClass, numberOfHosts
         );
         ipOutputLabel.setText(output);
     } catch (Exception ex) {
         ipOutputLabel.setText("An error occurred: " + ex.getMessage());
     }
 }

 private String generateSubnetMask(int prefixLength) {
     StringBuilder subnetMask = new StringBuilder();
     for (int i = 0; i < 32; i++) {
         subnetMask.append(i < prefixLength ? '1' : '0');
         if ((i + 1) % 8 == 0 && i != 31) {
             subnetMask.append('.');
         }
     }
     return subnetMask.toString();
 }

 private String generateSubnetMaskDecimal(int prefixLength) {
     int mask = 0xffffffff << (32 - prefixLength);
     return String.format("%d.%d.%d.%d",
             (mask >> 24) & 0xff,
             (mask >> 16) & 0xff,
             (mask >> 8) & 0xff,
             mask & 0xff);
 }

 private String calculateNetworkAddress(String binaryIP, String subnetMask) {
     StringBuilder networkAddress = new StringBuilder();
     for (int i = 0; i < binaryIP.length(); i++) {
         if (binaryIP.charAt(i) == '.' || subnetMask.charAt(i) == '.') {
             networkAddress.append('.');
         } else {
             networkAddress.append(binaryIP.charAt(i) == '1' && subnetMask.charAt(i) == '1' ? '1' : '0');
         }
     }
     return networkAddress.toString();
 }

 private String calculateNetworkAddressDecimal(String ipPart, String subnetMask) {
     String[] ipOctets = ipPart.split("\\.");
     String[] maskOctets = subnetMask.split("\\.");
     int[] networkAddress = new int[4];
     for (int i = 0; i < 4; i++) {
         networkAddress[i] = Integer.parseInt(ipOctets[i]) & Integer.parseInt(maskOctets[i]);
     }
     return String.format("%d.%d.%d.%d", networkAddress[0], networkAddress[1], networkAddress[2], networkAddress[3]);
 }

 private String calculateBroadcastAddress(String networkAddress, String subnetMask) {
     StringBuilder broadcastAddress = new StringBuilder();
     for (int i = 0; i < networkAddress.length(); i++) {
         if (networkAddress.charAt(i) == '.' || subnetMask.charAt(i) == '.') {
             broadcastAddress.append('.');
         } else {
             broadcastAddress.append(networkAddress.charAt(i) == '1' || subnetMask.charAt(i) == '0' ? '1' : '0');
         }
     }
     return broadcastAddress.toString();
 }

 private String calculateBroadcastAddressDecimal(String networkAddress, String subnetMask) {
     String[] networkOctets = networkAddress.split("\\.");
     String[] maskOctets = subnetMask.split("\\.");
     int[] broadcastAddress = new int[4];
     for (int i = 0; i < 4; i++) {
         broadcastAddress[i] = Integer.parseInt(networkOctets[i]) | ~Integer.parseInt(maskOctets[i]) & 0xff;
     }
     return String.format("%d.%d.%d.%d", broadcastAddress[0], broadcastAddress[1], broadcastAddress[2], broadcastAddress[3]);
 }

 private char determineClass(int firstOctet) {
     if (firstOctet >= 0 && firstOctet <= 127) {
         return 'A';
     } else if (firstOctet >= 128 && firstOctet <= 191) {
         return 'B';
     } else if (firstOctet >= 192 && firstOctet <= 223) {
         return 'C';
     } else if (firstOctet >= 224 && firstOctet <= 239) {
         return 'D';
     } else {
         return 'E';
     }
 }

 public static void main(String[] args) {
     new IPv4Details();
 }
}
```

### Parity Checker

```java
package data_communiation;

public class HammingCode extends JFrame {

    public HammingCode() {

        generateButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String input = inputArea.getText();
                String output = generateHammingCode(input);
                outputArea.setText(output);
            }
        });
    }

```

```java
   private String generateHammingCode(String input) {

      return "Generated Hamming Code"; // Placeholder

   }

   public static void main(String[] args) {
      new HammingCode().setVisible(true);
   }
}
```

### 2.4.1 Hamming Code Error Detection

```java

public void actionPerformed(ActionEvent e) {
      if (e.getSource() == checkButton) {
         String input = inputField.getText();
         if (input.isEmpty()) {
            errorLabel.setText("Invalid input: Input
  field is empty");
            correctButton.setVisible(false);
            correctedCodeLabel.setText("");
            return;
         } else if (input.length() != 7) {
            errorLabel.setText("We develop for exactly 7
  bits");
            correctButton.setVisible(false);
            correctedCodeLabel.setText("");
            return;
         }
         int errorPosition = checkErrorIndex(input);
         if (errorPosition > 0) {
            errorLabel.setText("Error position: " +
  errorPosition);
            correctButton.setVisible(true);
            correctedCodeLabel.setText("");
         } else {
            errorLabel.setText("Error: No");
            correctButton.setVisible(false);
            correctedCodeLabel.setText("");
         }
      } else if (e.getSource() == correctButton) {
         String input = inputField.getText();
         String correctedCodeword = correctError(input);
         correctedCodeLabel.setText("Corrected Codeword: "
   + correctedCodeword);
      }
   }

   private int checkErrorIndex(String input) {
        int[] rcwArray = new int[7];
        for (int i = 0; i < 7; i++) {
            rcwArray[i] = input.charAt(i) - '0';
        }

        int p1 = rcwArray[6] ^ rcwArray[4] ^ rcwArray[2] ^ rcwArray[0];
        int p2 = rcwArray[5] ^ rcwArray[4] ^ rcwArray[1] ^ rcwArray[0];
        int p4 = rcwArray[3] ^ rcwArray[2] ^ rcwArray[1] ^ rcwArray[0];
        int errorIndex = (p4 << 2) | (p2 << 1) | p1;

        return errorIndex;
    }

    private String correctError(String input) {
        int errorIndex = checkErrorIndex(input);
        StringBuilder correctedCodeword = new StringBuilder(input);

        if (errorIndex > 0) {
            int index = errorIndex - 1;
            correctedCodeword.setCharAt(index, correctedCodeword.charAt(index) == '0' ? '1' : '0');
        }

        return correctedCodeword.toString();
    }
```

# Chapter 3

# Performance Evaluation

## 3.1 Results Analysis/Testing

In this section, we thoroughly discuss the outcomes of our testing process. We carefully examine how well each part of the project performed, looking at factors like accuracy, speed, and ease of use. By testing everything extensively, we aim to see how effective each feature is and find any areas where we can make improvements. We also talk about how reliable our test results are and what they mean for the project as a whole. This section gives us a clear picture of how well the project is working and where we might need to make adjustments.

### 3.1.1 Home Page

The home page features four options:

1. Stuffing and De-stuffing
2. IPv4 Implementation
3. Parity Checker
4. Hamming Code Error Detection

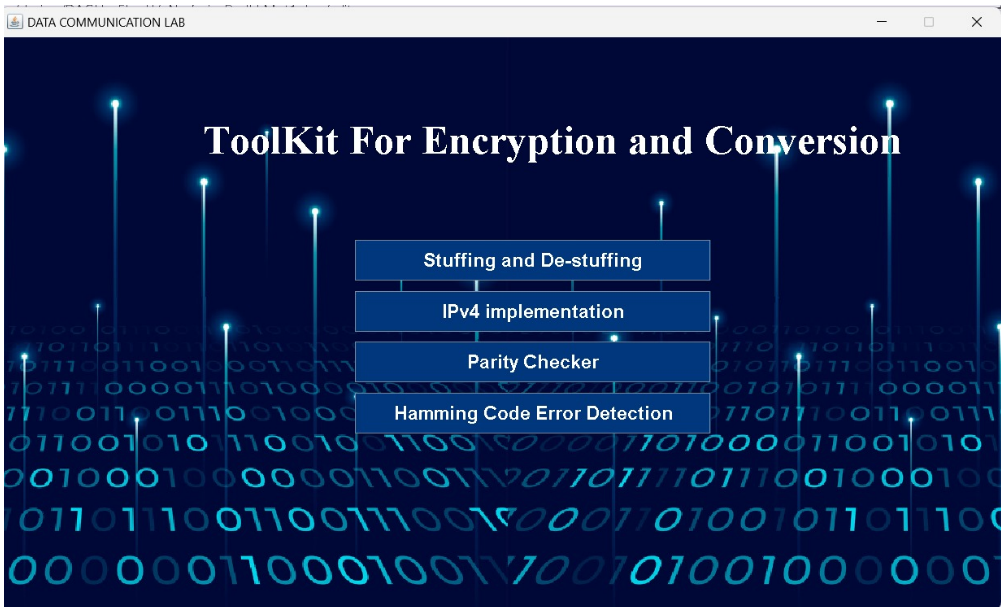


Figure 1: Home Page

### 3.1.2 Stuffing De-stuffing Menu

The Stuffing De-stuffing menu provides options for various stuffing and de-stuffing functionalities.

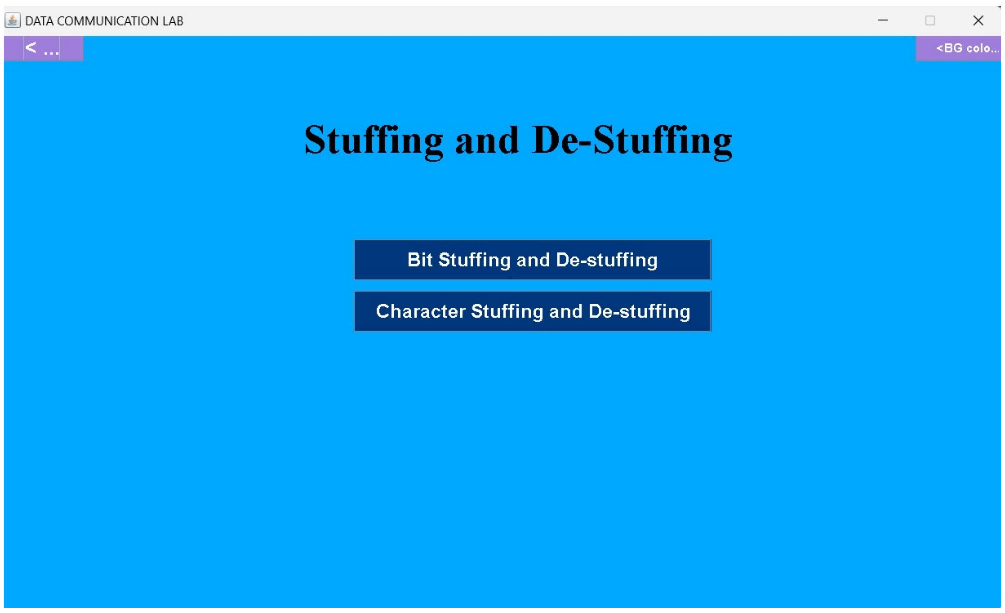
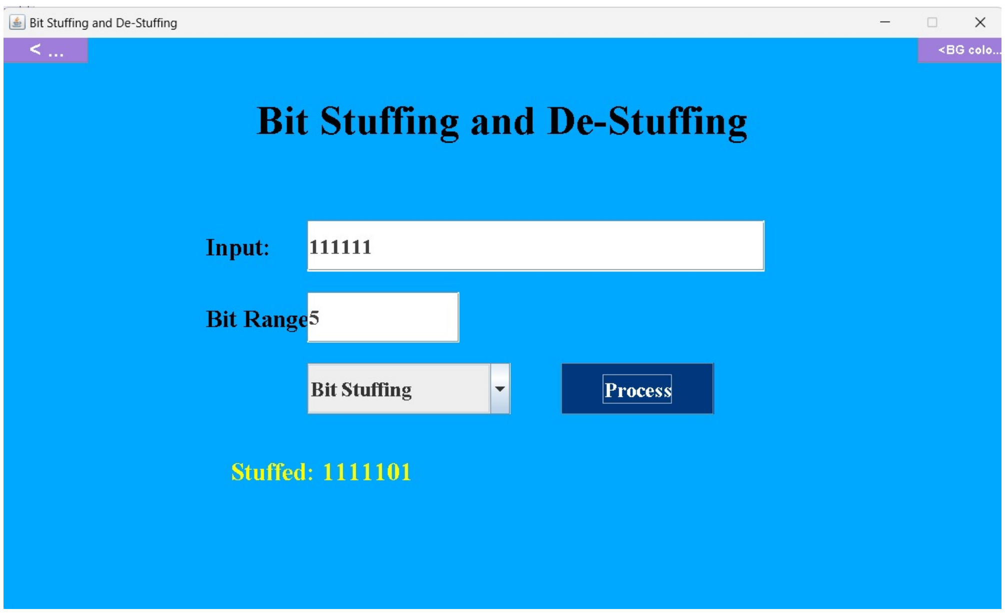

DATA COMMUNICATION LAB
< ...
<BG colo...
Stuffing and De-Stuffing
Bit Stuffing and De-stuffing
Character Stuffing and De-stuffing

Figure 2: Stuffing De-stuffing Menu

### 3.1.3 Bit Stuffing and De-stuffing Output

In this section, users input the bits and specify the bit range where stuffing is to be applied.



Bit Stuffing and De-Stuffing

Input: 111111

Bit Range 5

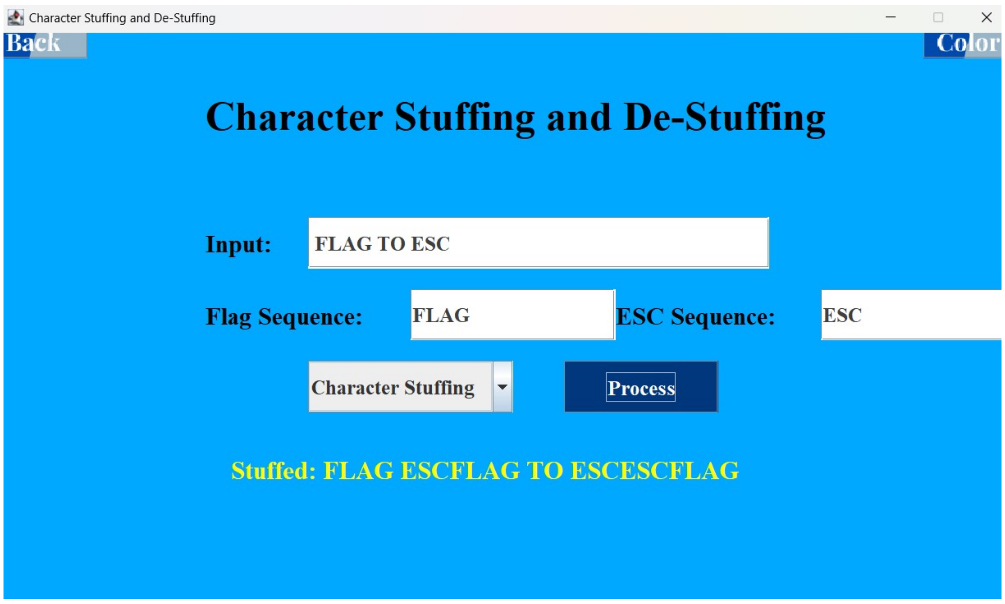
Bit Stuffing

Stuffed: 1111101

Figure 3: Bit Stuffing Input and Output

### 3.1.4 Character Stuffing and De-stuffing

Users provide the flag and escape values along with the input string for character stuffing and de-stuffing.



Character Stuffing and De-Stuffing

Input: FLAG TO ESC

Flag Sequence: FLAG ESC Sequence: ESC

Character Stuffing

Stuffed: FLAG ESCFLAG TO ESCESCFLAG

Figure 4: Character Stuffing and De-stuffing with Input and Output

### 3.1.5 IPv4 Menu

This section provides users with two options: IPv4 Conversion and IPv4 Details.

Users can choose between these options based on their specific needs, making the interface user-friendly and intuitive.

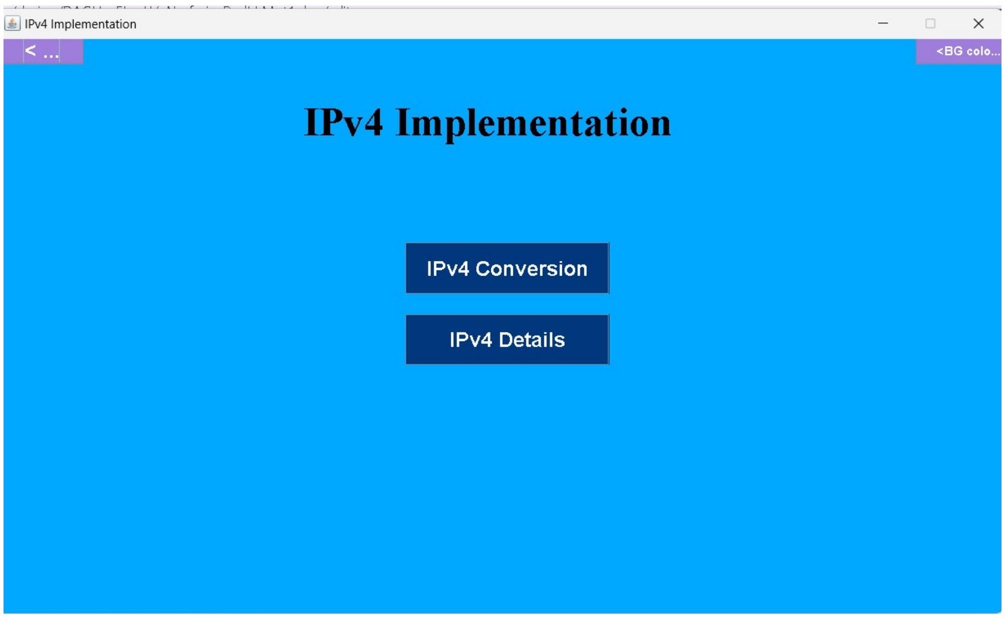
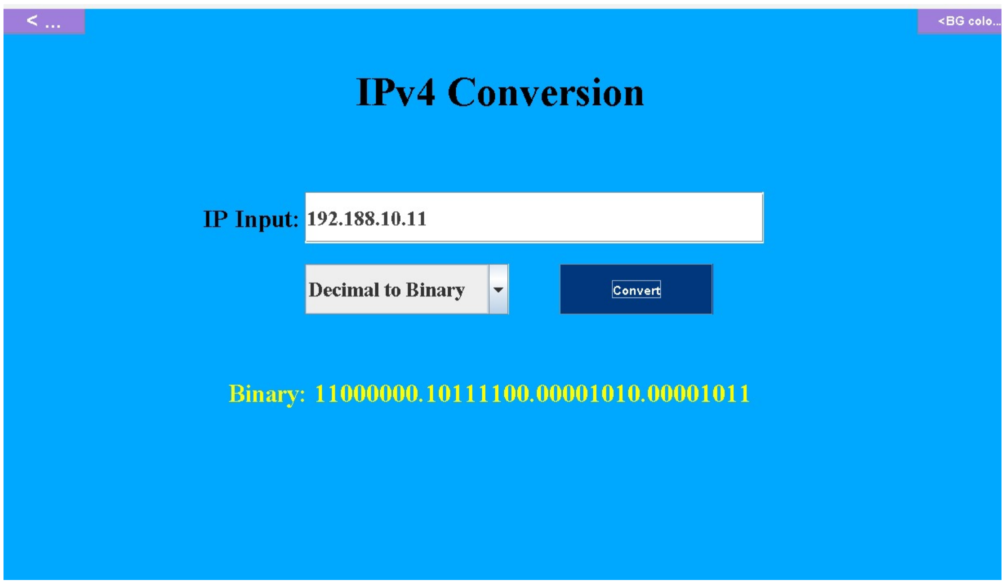


Figure 5: IPv4 Menu

### 3.1.6 IPv4 Conversion

This subsection facilitates the conversion of binary to decimal and decimal to binary representations of IPv4 addresses.

It is a crucial functionality for network administrators and developers to manipulate and understand IP addresses efficiently.



IP Input: 192.188.10.11

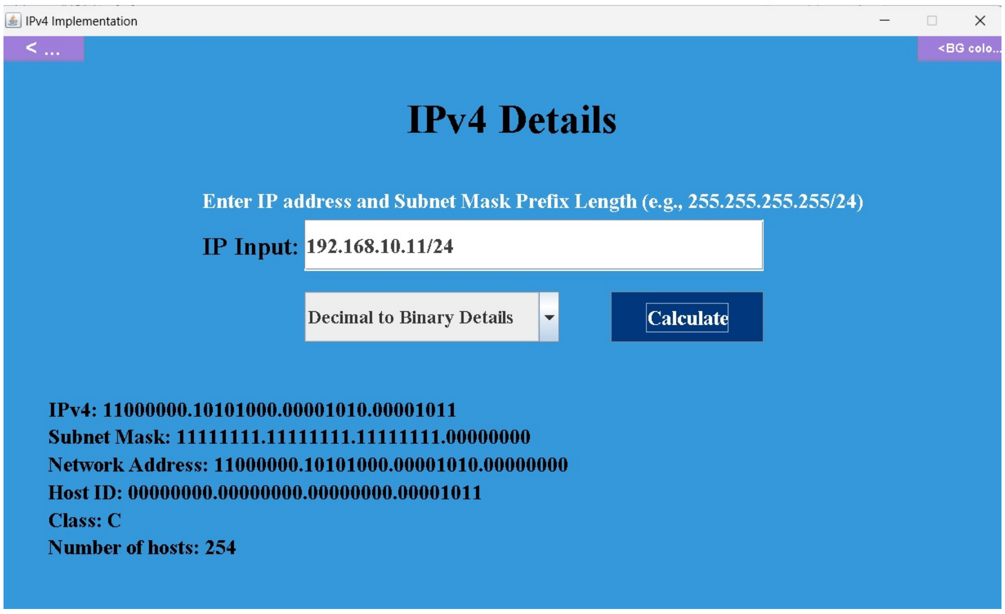
Binary: 11000000.10111100.00001010.00001011

Figure 6: IPv4 Conversion with Input and Output

### 3.1.7 IPv4 Details

This subsection provides details such as subnet mask, network address, broadcast address, class, and number of hosts.

It is essential for network administrators to have access to this information for proper network management and troubleshooting.


IPv4 Implementation
< ...
<BG colo...
IPv4 Details
Enter IP address and Subnet Mask Prefix Length (e.g., 255.255.255.255/24)
Decimal to Binary Details
Calculate

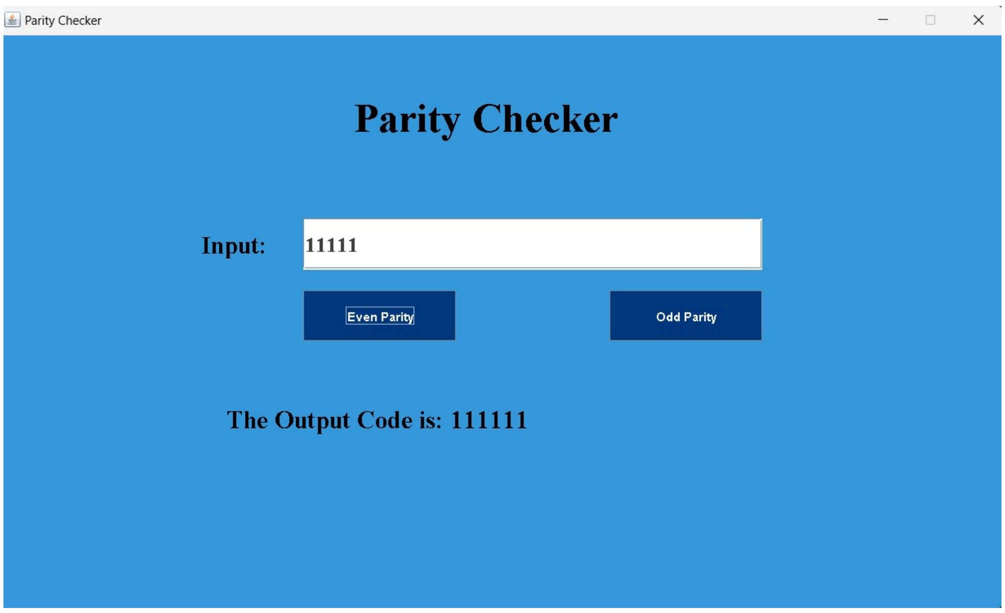
IP Input: 192.168.10.11/24

```
IPv4: 11000000.10101000.00001010.00001011
Subnet Mask: 11111111.11111111.11111111.00000000
Network Address: 11000000.10101000.00001010.00000000
Host ID: 00000000.00000000.00000000.00001011
Class: C
Number of hosts: 254
```

Figure 7: IPv4 Details with Input and Output

### 3.1.8 Parity Checker Code

This section implements the Hamming code parity checker algorithm. It follows the principle of even parity, where if the number of bits is even, a parity bit of 0 is added, and if the number of bits is odd, a parity bit of 1 is added.



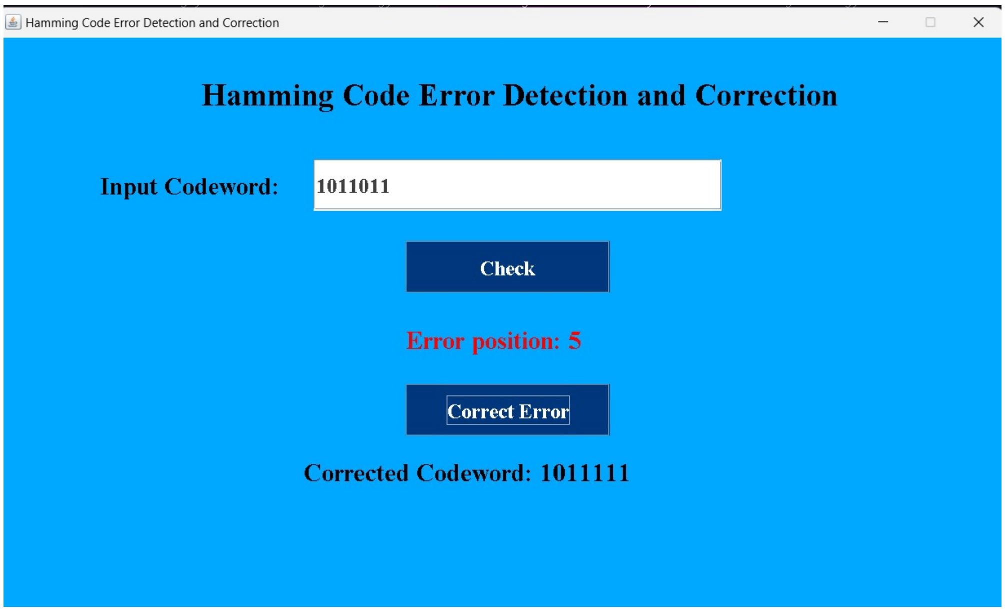
Input: 11111

The Output Code is: 111111

Figure 8: Hamming Code with Input and Output

### 3.1.9 Hamming Code Error Detection

In this section, the system detects errors in the Hamming code. If any errors are found, the system automatically corrects them.


Hamming Code Error Detection and Correction

Hamming Code Error Detection and Correction

Input Codeword: 1011011

Check

Error position: 5

Correct Error

Corrected Codeword: 1011111

Figure 9: Hamming Code Error Detection and Correction

### 3.1.10 Color Change

We can change the background color at any time by clicking the color button.

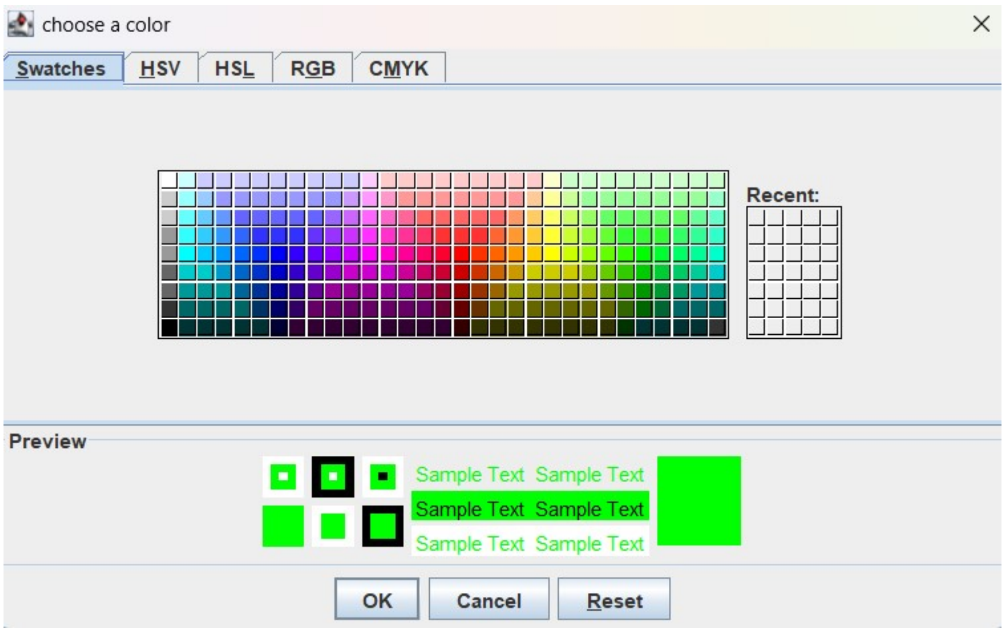


Figure 10: Background color change

## 3.2 Results Overall Discussion

In this chapter, we talk about the results of our project. We discuss what we achieved and what problems we faced along the way. We look at how our results match up with what we wanted to achieve initially. We also discuss any surprises or unexpected findings we encountered. This discussion helps us understand the impact of our work and highlights areas where we can improve or explore further. Overall, it gives us a good sense of where our project stands and what we can learn from it.

### 3.2.1 Complex Engineering Problem Discussion

Our project deals with a complex engineering challenge, touching upon several crucial aspects. We have carefully addressed each of these aspects, employing effective strategies to overcome them. This subsection delves into the details of these challenges and how we tackled them.

# Chapter 4

# Conclusion

## 4.1 Discussion

In this chapter, we delve into a comprehensive discussion regarding the contents presented earlier. We summarize the essence of our work, encompassing its description, results, and observations. Throughout the project, we aimed to develop a versatile toolkit for data communication, offering various functionalities such as IPv4 implementation, stuffing and de-stuffing techniques, and Hamming code error detection and correction. Through meticulous implementation and testing, we successfully achieved our objectives, ensuring the reliability and effectiveness of each component. Our observations reveal the robustness of the toolkit, its user-friendly interface, and its potential for further enhancements. Overall, this project demonstrates our commitment to addressing key challenges in data communication while providing valuable insights for future development and research endeavors.

## 4.2 Limitations

The project, despite its achievements, is not without its limitations. One notable limitation lies in the scope of functionalities offered by the toolkit. While it covers essential aspects such as IPv4 implementation, stuffing and de-stuffing techniques, and Hamming code error detection and correction, it may lack some advanced features found in more specialized software. Additionally, the project's reliance on specific algorithms and methods may limit its applicability to certain scenarios or data communication protocols. Furthermore, the toolkit's user interface, although designed to be intuitive, may pose challenges for users unfamiliar with the concepts of data communication. Lastly, due to time and resource constraints, the project may not have undergone extensive testing across various environments, potentially leading to undiscovered bugs or compatibility issues. Despite these limitations, the project serves as a valuable starting point for future improvements and expansions in the field of data communication

## 4.3 Scope of Future Work

Looking ahead, there are several areas where the project can grow and improve:

1. **Feature Expansion:**We aim to add more features to enhance what the project can do, making it more useful and versatile.

2. **Advanced Techniques:**We plan to incorporate advanced methods and algorithms to tackle more complex problems effectively.

3. **User Interface Improvement:** Our goal is to make the project easier to use by refining its design and making it more intuitive for users.

4. **Platform Expansion:**We'll explore adapting the project for different platforms like mobile devices or cloud services, making it more accessible to a wider audience.

5. **Security Boost:** We aim to strengthen security measures to ensure data safety and privacy, implementing stronger encryption and authentication methods.

# References

1: GeeksforGeeks

2: Neso Academy

3: Report Overleaf