PASSWORD GENERATOR - REPORT

Submitted by

50 – Shubham Bakade

59 – Sahil Mahadik

62 – Sahil Raut

63 – Sarfaraz Shaik

Under the Guidance of………

PROF. Mansi Mohite

DEPARTMENT OF
ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

KONKAN GYANPEETH COLLEGE ENGINEERING KARJAT-410201

(2024-2025)

CERTIFICATE

This to verify the project entitled "**Password Generator**" is a bonafide work of "**Shubham Bakade**, **Sahil Mahadik**, **Sahil Raut**, **Sarfaraz Shaik**" submitted to University of Mumbai in partial fulfilment of the requirement of the award of the degree of "**S.E in Artificial Intelligence & Data Science**"

Project Guide

PROF. MANSI MOHITE

DEPARTMENT OF
ARTIFICIAL INTELLIGENCE AND DATA SCIENCE
KONKAN GYANPEETH COLLEGE ENGINEERING KARJAT - 410201

ABSTRACT

This Java project involves the development of a *Strong Password Generator* using the Swing GUI framework. The application allows users to generate strong passwords based on their chosen criteria, including password length and the inclusion of uppercase letters, lowercase letters, numbers, and special characters. A SecureRandom function is employed to ensure randomness and security of the generated passwords.

ACKNOWLEDGEMENT

We express gratitude to the open-source Java development community for providing extensive resources and documentation, making it easier to implement secure and efficient password generation. Special thanks go to the developers of the javax.swing library and the SecureRandom API, which played a critical role in the implementation of this project.

INDEX

INTRODUCTION

In today's digital era, secure passwords are crucial for protecting personal information. A weak password is highly susceptible to hacking attempts, making it necessary to generate strong passwords. This project provides a graphical password generator where users can define various password characteristics such as length, inclusion of uppercase and lowercase letters, numbers, and special characters. The program offers a user-friendly interface to meet these security needs effectively.

THEORY

This project integrates two core principles: **randomness** for secure password generation and **graphical user interface (GUI) development** using Java's `Swing` framework. The randomness is achieved through the use of `SecureRandom`, a class from the `java.security` package, which is specifically designed to generate cryptographically strong random numbers. This ensures that the generated passwords are not only random but also resistant to predictable patterns or attacks. By leveraging `SecureRandom`, the application enhances security compared to traditional random number generators like `Math.random()`, which are not suitable for cryptographic purposes.

The `Swing` library, part of `javax.swing`, provides the foundation for the graphical user interface (GUI). Swing is widely used in Java for building rich client-side applications due to its ease of use, flexibility, and ability to create complex interfaces with relatively little code. In this project, Swing components like `JTextField`, `JCheckBox`, `JButton`, and `JPanel` are used to allow users to interact with the application by selecting password preferences. These preferences include the length of the password and the types of characters to include, such as uppercase letters, lowercase letters, numbers, and special characters. By providing checkboxes, the interface enables users to customize their password based on specific security needs.

The password generation process begins by assembling a character set based on the user's selections. If the user selects multiple character types (e.g., both uppercase and lowercase letters, numbers, and special characters), the character set becomes more diverse, leading to a stronger password. The strength of a password is directly related to its length and the variety of characters it contains. A longer password that includes multiple character types is significantly more resistant to brute-force attacks, where an attacker attempts to guess a password by systematically trying every possible combination.

Randomness plays a vital role in preventing attackers from predicting password patterns. If passwords were generated in a predictable manner, it would be easier for malicious actors to crack them. This is why `SecureRandom` is used instead of simpler random generators. By generating a sequence of random characters from the selected character set, the application ensures that each password is unique and difficult to guess, providing a strong defense against common attack methods like dictionary attacks or pattern recognition.

OPERATION AND FUNCTIONING

The Strong Password Generator application follows these steps:

1. The user inputs a desired password length and selects the types of characters to include: uppercase, lowercase, numbers, and/or special characters.

2. Upon clicking "Generate Password," the application collects the user's preferences and constructs a character set.

3. A SecureRandom instance is used to randomly select characters from this set, and a password of the specified length is created.

4. The generated password is then displayed in a non-editable text field.

5. The user can click "Reset" to clear the fields and make a new selection for password generation.

SYSTEM DESIGN

The system is designed using Java Swing components for the GUI and the SecureRandom API for secure password generation:

- **GUI Components:**
  - JTextField: Used for entering the password length and displaying the generated password.
  - JCheckBox: Allows users to select the types of characters to include in the password (uppercase, lowercase, numbers, special characters).
  - JButton: Provides buttons for generating and resetting the password.
  - JPanel: Organizes the layout of various components.

- **Backend Logic:**
  - The program uses the SecureRandom class for cryptographic-strength random number generation.
  - Based on the selected checkboxes, a character set is dynamically built, and a password of the specified length is generated from this set.
  - Input validation is implemented to ensure that the length field contains a valid integer and that at least one character set is selected.

REQUIREMENTS

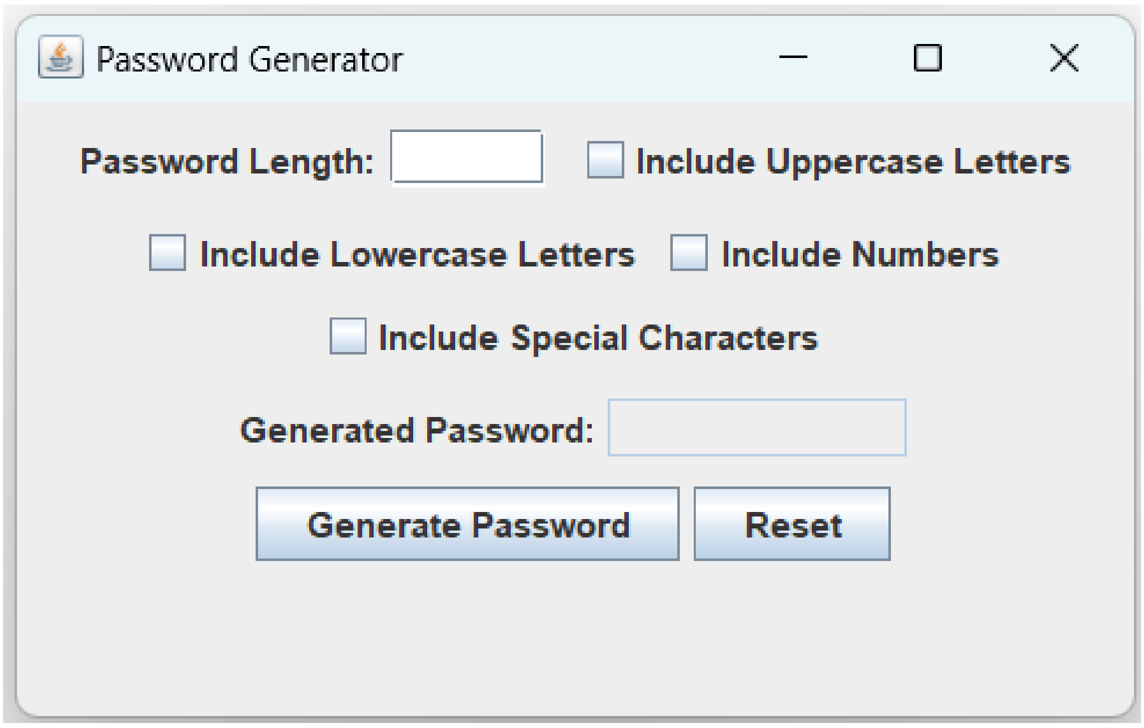
- **Software:**
  - Java Development Kit (JDK) 8 or higher.
  - Any IDE that supports Java (Eclipse, IntelliJ IDEA, etc.).
- **Libraries:**
  - javax.swing for GUI components.
  - java.security.SecureRandom for password generation.
- **Functional Requirements:**
  - The user should be able to select the password length.
  - The user should be able to choose character types (uppercase, lowercase, numbers, special characters).
  - The generated password must be displayed in a non-editable field.
  - Validation should ensure that valid input is provided.

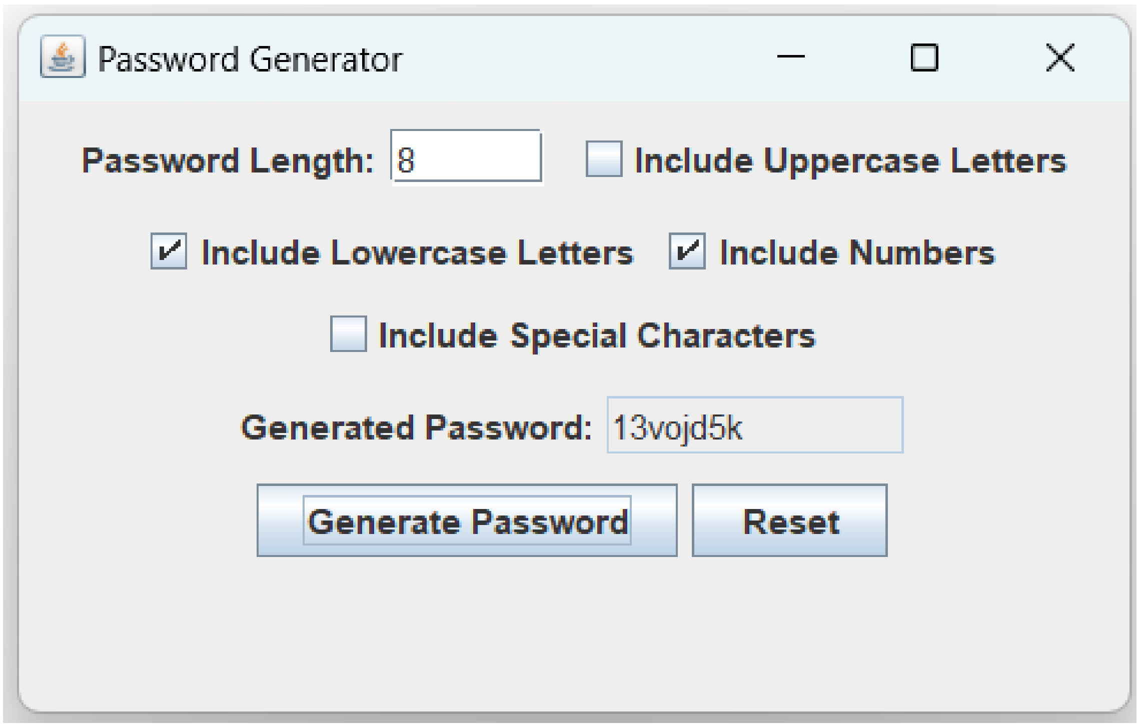
IMPLEMENTATION

CODE:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.security.SecureRandom;

public class StrongPasswordGenerator extends JFrame {
    // Character sets to be used in password generation
    private static final String UPPERCASE = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private static final String LOWERCASE = "abcdefghijklmnopqrstuvwxyz";
    private static final String NUMBERS = "0123456789";
    private static final String SPECIAL_CHARACTERS = "!@#$%^&*()-_=+[]{}|;:',.<>?/";

    private JTextField lengthField;
    private JCheckBox uppercaseCheck;
    private JCheckBox lowercaseCheck;
    private JCheckBox numbersCheck;
    private JCheckBox specialCheck;
    private JTextField passwordField;
    private JButton generateButton;
    private JButton resetButton;

    public StrongPasswordGenerator() {
        setTitle("Password Generator");
        setSize(400, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        // Components for password length
        JPanel lengthPanel = new JPanel();
        lengthPanel.add(new JLabel("Password Length:"));
        lengthField = new JTextField(5);
        lengthPanel.add(lengthField);
        add(lengthPanel);

        // Checkboxes for character types
        uppercaseCheck = new JCheckBox("Include Uppercase Letters");
        lowercaseCheck = new JCheckBox("Include Lowercase Letters");
        numbersCheck = new JCheckBox("Include Numbers");
        specialCheck = new JCheckBox("Include Special Characters");
        add(uppercaseCheck);
        add(lowercaseCheck);
        add(numbersCheck);
        add(specialCheck);
        // Panel for generated password
        JPanel passwordPanel = new JPanel();
        passwordPanel.add(new JLabel("Generated Password:"));
        passwordField = new JTextField(10);
        passwordField.setEditable(false);
        passwordPanel.add(passwordField);
        add(passwordPanel);

        // Button to generate the password
        generateButton = new JButton("Generate Password");
        generateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                generateAndDisplayPassword();
            }
        });
        add(generateButton);

        // Add Reset button to reset fields
        resetButton = new JButton("Reset");
        resetButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                resetFields();
            }
        });
        add(resetButton);
    }

    // Method to generate and display the password
    private void generateAndDisplayPassword() {
        try {
            int length = Integer.parseInt(lengthField.getText());
            boolean includeUppercase = uppercaseCheck.isSelected();
            boolean includeLowercase = lowercaseCheck.isSelected();
            boolean includeNumbers = numbersCheck.isSelected();
            boolean includeSpecial = specialCheck.isSelected();

            String password = generatePassword(length, includeUppercase, includeLowercase, includeNumbers,
includeSpecial);
            passwordField.setText(password);
        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Please enter a valid number for the password length.",
"Error", JOptionPane.ERROR_MESSAGE);
        } catch (IllegalArgumentException e) {
            JOptionPane.showMessageDialog(this, e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    // Method to generate a password based on user preferences
    private String generatePassword(int length, boolean includeUppercase, boolean includeLowercase,
boolean includeNumbers, boolean includeSpecial) {
        String characterSet = "";
        // Build character set based on user choices
        if (includeUppercase) characterSet += UPPERCASE;
        if (includeLowercase) characterSet += LOWERCASE;
        if (includeNumbers) characterSet += NUMBERS;
        if (includeSpecial) characterSet += SPECIAL_CHARACTERS;



        // If no character sets are selected, throw an exception
        if (characterSet.isEmpty()) {
            throw new IllegalArgumentException("At least one character set must be selected.");
        }

        SecureRandom random = new SecureRandom();
        StringBuilder password = new StringBuilder(length);

        // Generate random password from the chosen character set
        for (int i = 0; i < length; i++) {
            int index = random.nextInt(characterSet.length());
            password.append(characterSet.charAt(index));
        }

        return password.toString();
    }

    // Method to reset all fields
    private void resetFields() {
        lengthField.setText("");
        passwordField.setText("");
        uppercaseCheck.setSelected(false);
        lowercaseCheck.setSelected(false);
        numbersCheck.setSelected(false);
        specialCheck.setSelected(false);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            StrongPasswordGenerator gui = new StrongPasswordGenerator();
            gui.setVisible(true);
        });
    }
}
```

OUTPUT:




Password Length: 8

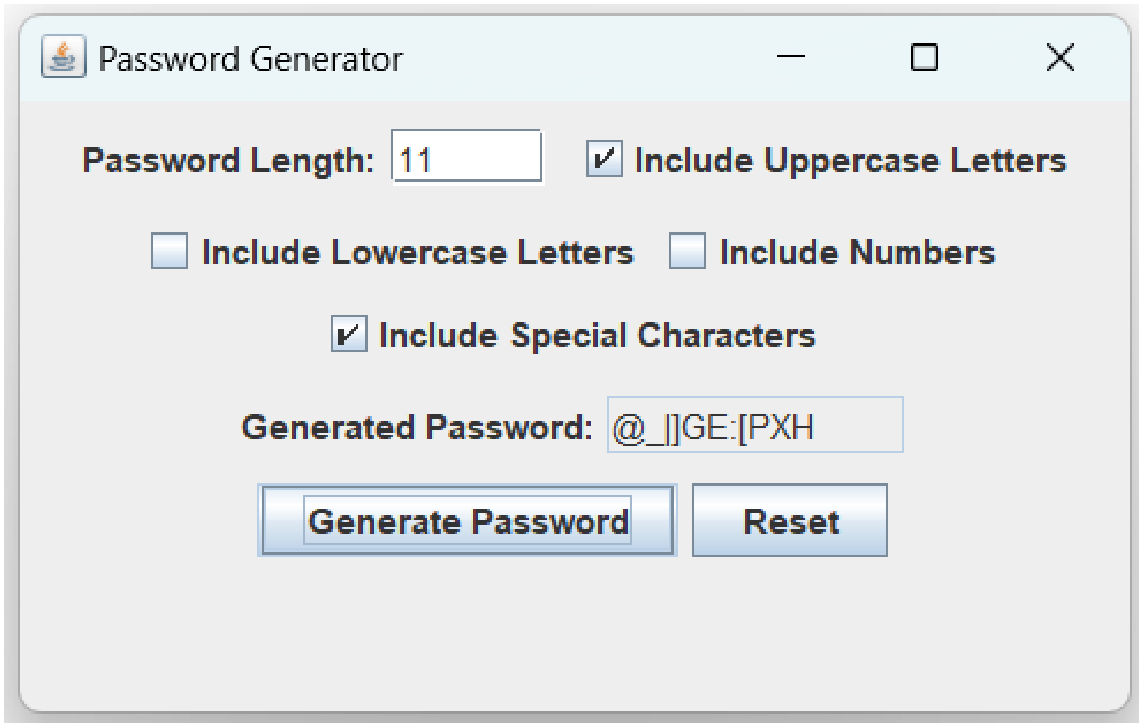
☐ Include Uppercase Letters

✓ Include Lowercase Letters

✓ Include Numbers

☐ Include Special Characters

Generated Password: 13vojd5k


Password Generator

Password Length: 11

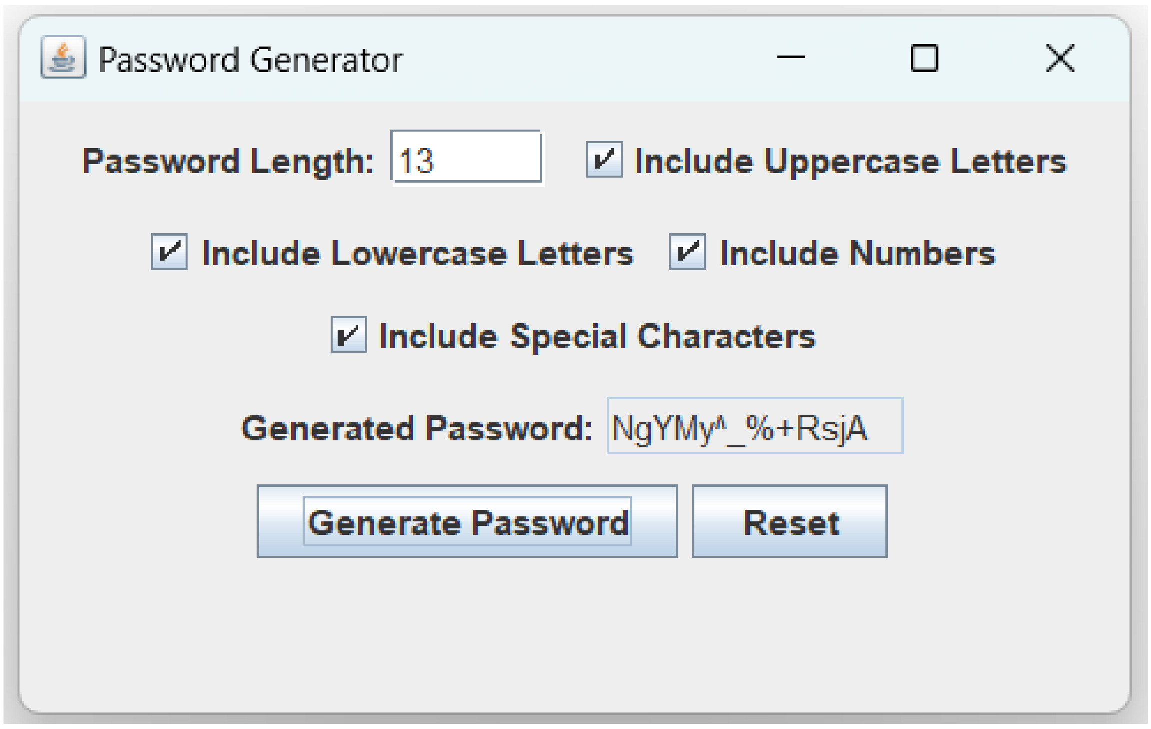
[✓] Include Uppercase Letters

[ ] Include Lowercase Letters

[ ] Include Numbers

[✓] Include Special Characters

Generated Password: @_|]GE:[PXH


Password Generator

Password Length: 13

[✓] Include Uppercase Letters

[✓] Include Lowercase Letters

[✓] Include Numbers

[✓] Include Special Characters

Generated Password: NgYMy^_%+RsjA

CONCLUSION

This project demonstrates the practical implementation of a secure password generator using Java. It highlights the use of SecureRandom for generating cryptographically strong passwords and the simplicity of creating GUI applications using Swing. By offering a customizable password creation tool, the application meets modern security needs in an accessible and user-friendly way. Users can easily create strong passwords with the flexibility to choose their own security preferences.

REFERENCES

1. ava API Documentation:
   https://docs.oracle.com/javase/8/docs/api/

2. Swing GUI Tutorial:
   https://docs.oracle.com/javase/tutorial/uiswing/

3. SecureRandom Class:
   https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html

4. Oracle Java Secure Coding Guidelines:
   https://www.oracle.com/java/technologies/javase/seccodeguide.html