### **CAPTCHA GENERATOR**

### 

### **Table of Contents**

### **●Abstract**

### **●Introduction**

### **●System Requirements**

### **●Design and Architecture**

### **●Implementation of Code**

### **●Conclusion**

### 

### **Abstract**

This document presents the development of a CAPTCHA generation system using Java. The system is designed to create CAPTCHAs to prevent automated bots from accessing web services. It explores the fundamental principles of CAPTCHA, the design considerations, and the technical implementation in Java. The aim is to provide a robust and user-friendly solution that enhances security for web applications.

### **Introduction**

CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart) are widely used to differentiate between human users and automated bots. They are crucial for protecting online services from abuse and spam. This project focuses on building a CAPTCHA generation system using Java. The system generates various types of CAPTCHAs, including text-based and image-based, to ensure robust security. This document outlines the system requirements, design and architecture, implementation, and concludes with an evaluation of the solution.

### **System Requirements**

**Hardware Requirements:**

* Processor: Intel i3 or higher
* RAM: 4 GB or more
* Storage: 500 MB of available disk space

**Software Requirements:**

* Operating System: Windows, macOS, or Linux
* Java Development Kit (JDK) 8 or higher
* Integrated Development Environment (IDE) such as IntelliJ IDEA, Eclipse, or NetBeans
* Libraries: Java AWT, Java Swing, and additional libraries for image processing (e.g., Java ImageIO)

### **Design and Architecture**

The design of the CAPTCHA generation system involves the following components:

1. **User Interface:** A simple GUI to display CAPTCHAs and accept user input.
2. **CAPTCHA Generator:** A core component responsible for generating different types of CAPTCHAs. This includes:
   * Text-based CAPTCHAs: Randomly generated strings with distortion.
   * Image-based CAPTCHAs: Randomly generated images with various patterns.
3. **Validation Module:** A module to verify if the user's input matches the generated CAPTCHA.
4. **Storage Module:** An optional component to store generated CAPTCHAs and user responses for analysis.

The architecture is modular, allowing easy extension and maintenance. The system follows the Model-View-Controller (MVC) design pattern to separate concerns and improve scalability.

**Implementation of Code**

The implementation is divided into the following steps:

●Setup Environment:

•Install JDK and set up the IDE.

•Import necessary libraries for GUI and image processing.

●CAPTCHA Generator Class:

StringBuilder();

for (int i = 0; i < CAPTCHA\_LENGTH; i++) {

captchaText.append(CHARACTERS.charAt(random.nextInt(CHARACTERS.length())));

}

return captchaText.toString();

}

public BufferedImage generateCaptchaImage(String captchaText) {

int width = 160;

int height = 60;

BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

Graphics2D g2d = image.createGraphics();

Random random = new Random();

// Fill background

g2d.setColor(Color.WHITE);

g2d.fillRect(0, 0, width, height);

// Draw text with random colour and distortions

for (int i = 0; i < captchaText.length(); i++) {

g2d.setColor(new Color(random.nextInt(256), random.nextInt(256), random.nextInt(256)));

g2d.drawString(String.valueOf(captchaText.charAt(i)), 20 \* i + 15, 45 + random.nextInt(10));

}

g2d.dispose();

return image;

}

}

●User Interface Class:

import javax.swing.\*;

import java.awt.\*;

import java.awt.image.BufferedImage;

public class CaptchaUI extends JFrame {

private CaptchaGenerator captchaGenerator;

private String currentCaptchaText;

public CaptchaUI() {

captchaGenerator = new CaptchaGenerator();

initUI();

}

private void initUI() {

setTitle("CAPTCHA Generator");

setSize(300, 200);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

currentCaptchaText = captchaGenerator.generateCaptchaText();

BufferedImage captchaImage = captchaGenerator.generateCaptchaImage(currentCaptchaText);

JLabel captchaLabel = new JLabel(new ImageIcon(captchaImage));

JTextField captchaInput = new JTextField(10);

JButton submitButton = new JButton("Submit");

submitButton.addActionListener(e -> {

String userInput = captchaInput.getText();

if (userInput.equals(currentCaptchaText)) {

JOptionPane.showMessageDialog(this, "CAPTCHA Verified!");

} else {

JOptionPane.showMessageDialog(this, "Incorrect CAPTCHA. Try again.");

}

currentCaptchaText = captchaGenerator.generateCaptchaText();

captchaLabel.setIcon(new ImageIcon(captchaGenerator.generateCaptchaImage(currentCaptchaText)));

});

setLayout(new FlowLayout());

add(captchaLabel);

add(captchaInput);

add(submitButton);

}

public static void main(String[] args) {

EventQueue.invokeLater(() -> {

CaptchaUI ex = new CaptchaUI();

ex.setVisible(true);

}

}

}

Conclusion

The CAPTCHA generation system developed using Java provides an effective solution to enhance security for web applications by preventing automated bots from accessing services. The modular design and implementation ensure scalability and maintainability, allowing for easy extension to include more advanced CAPTCHA types. By using standard Java libraries and adhering to the MVC design pattern, the system is both robust and user-friendly. Future enhancements could include integrating more complex CAPTCHA types and improving the user interface for better usability.