## SAFETYNET: IMAGE-BASED RISK ASSESSMENT API USING JAVA SWING LIBRARY

Project

*Submitted by*

*CH.EN.U4CYS22018 HARISH MADHAVAN J U*

*CH.EN.U4CYS22034 NAREN S*

*CH.EN.U4CYS22043 R SUBRAMANIAN*

*For the course 20CYS383 Java Programming Lab*

IV Semester (2022-2023)

B. Tech in Computer Science and Engineering (Cyber Security)

SAFETYNET: IMAGE-BASED RISK ASSESSMENT API USING JAVA SWING LIBRARY

|  |  |
| --- | --- |
| S.NO | CONTENTS |
| 1 | ABSTRACT |
| 2 | INTRODUCTION |
| 3 | PROJECT DESCRIPTION |
| 4 | METHODOLOGY |
| 5 | SCHEMATIC DIAGRAM |
| 6 | SOURCE CODE |
| 7 | IMPLEMENTATION |
| 8 | OUTPUT |
| 9 | CONCLUSION |

# Abstract

Purpose: The primary purpose of this Java application is to analyze images provided by users, focusing specifically on identifying potential weapons within those images.

User Interface: The application utilizes Swing, a Java GUI toolkit, to create a user-friendly interface. Swing provides a set of GUI components like buttons, text fields, and panels that allow for the creation of interactive graphical applications.

Image Analysis: The core functionality of the application involves analyzing the user-provided images. This analysis is aimed at detecting potential weapons within the images.

Report Generation: After analyzing the image, the application generates informative reports. These reports include various details about the image and its analysis:

Violent Percentage: This indicates the application's certainty level in its findings regarding potential weapons. It's essentially a measure of confidence in the analysis results.

Processing Time: The time taken by the application to process the image and perform the analysis.

Basic Image Attributes: Details such as height and width of the image are included in the report.

Weapon Classification: The identified weapons are categorized into a predefined class system. For example, firearms might be one category, while bladed weapons could be another.

Unique Weapon ID: Each identified weapon is assigned a unique identifier for reference purposes. This allows for easy tracking and referencing of specific weapons identified in the image.

Predefined Class System: The application uses a predefined classification system to categorize the identified weapons. This system likely includes various categories such as firearms, bladed weapons, explosives, etc. Categorizing the weapons helps organize the information and provides a structured way to understand the types of weapons detected.

Overall, this Java application provides a user-friendly interface for analyzing images, focusing specifically on identifying potential weapons. It generates detailed reports to provide insights into the analysis results, including confidence levels, processing times, and categorized information about the detected weapons.

# Introduction

In today's digital world, image processing plays a pivotal role in various domains, such as computer vision, medical imaging, and multimedia applications. It encompasses a multitude of techniques designed to extract valuable information from images. Weapon identification is a crucial aspect of security applications. This project aims to develop a user-friendly tool that assists in identifying weapons within images. The application caters to users who require a basic weapon identification system

User Interface (UI):

* Utilizes Swing framework for a user-friendly interface.
* Allows image upload and analysis initiation.

Image Analysis:

* Employs image processing to detect potential weapons.
* May use machine learning for accuracy.

Report Generation:

* Generates reports with key metrics.
* Includes Violent Percentage, processing time, and basic image attributes.
* Categorizes weapons and assigns unique IDs.

Predefined Class System:

* Organizes identified weapons into categories (e.g., firearms, bladed weapons).

Implementation Considerations:

* May use third-party libraries for processing.
* Requires error handling, input validation, and performance optimization.
* Considers user privacy and security.

Overall, the Java application offers a user-friendly interface for analyzing uploaded images, detecting weapons, and generating detailed reports.

# Project Description

**1. User Interface**

* The application utilizes Swing, a Java GUI framework, to create a user-friendly interface.
* Users can seamlessly input images through a file selection mechanism.

Clear instructions and labels guide users through the image analysis process.

**2. Image Processing**

* The application employs image processing techniques to extract relevant features from the user-provided image.
* This may involve operations like colour space conversion, or filtering, depending on the chosen approach.

While not utilizing machine learning, the application might leverage pre-defined rules or patterns to identify potential weapons based on extracted features.

**3. Report Generation**

Once the image analysis is complete, the application generates a comprehensive report that includes:

* Violent percentage: A numerical value reflecting the algorithm's certainty in the extracted information.
* Time Taken: The elapsed time during image processing, providing performance insights.
* Image Description: Precise dimensions (height and width) of the image.
* Weapon Category: the recognized Weapon category.
* Weapon ID: A unique identifier associated with the assigned Weapon, facilitating further analysis.

**4. Output and Visualization**

* The generated report is presented to the user in a clear and concise format, potentially using Swing elements like text boxes or tables.
* The application may also choose to visually highlight potential weapons within the analysed image for enhanced user comprehension.

# Methodology

**1. User Interface Creation (JFrame and Swing Components):**

* The code creates a JFrame named "Image Inference" with a set size and default close operation.
* It defines a JButton named "Select Image" for user interaction.
* A JTextArea named resultTextArea is created to display the inference results. This area is non-editable and has a set font style and size.
* A JScrollPane is used to provide scrollable functionality for the results text area within the limited window space.
* A JPanel is created with a BorderLayout layout manager.
* The button and scroll pane are added to the panel at specific locations (North and Center) using the layout manager.
* Finally, the panel is added to the main JFrame window.

**2. User Interaction and Image Selection:**

* An ActionListener is attached to the "Select Image" button.
* When the button is clicked, a JFileChooser dialog appears, allowing the user to select an image file.
* If the user clicks "Open" (JFileChooser.APPROVE\_OPTION), the selected file's path is retrieved.
* The processImage method is called to handle the selected image.

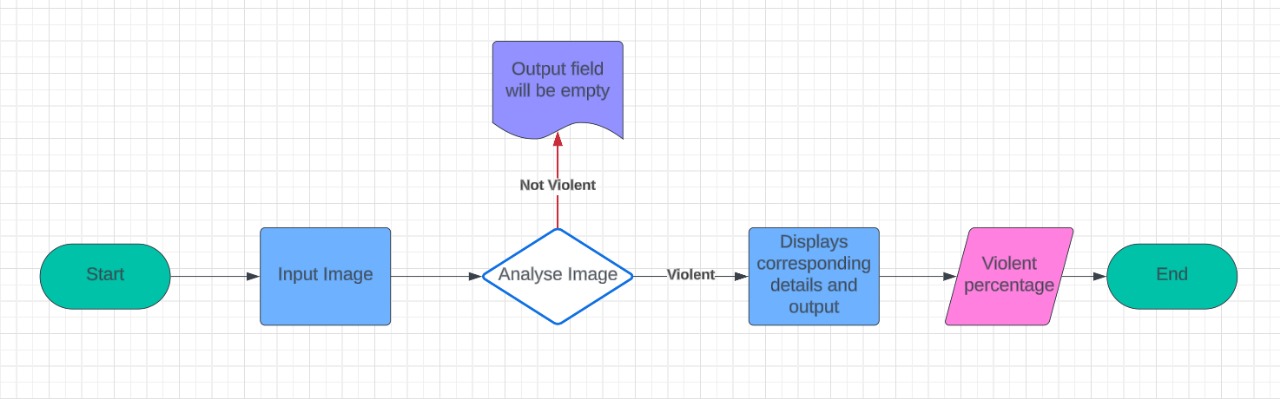
**3. Image Processing and API Interaction:**

* The processImage method takes the selected image path as input.
* It reads the entire image file into a byte array using Files.readAllBytes.
* The byte array is then encoded into a Base64 string using Base64.getEncoder().encodeToString.
* The code defines an API key and model endpoint (likely specific to Roboflow).
* It constructs a URL for the inference API call, including the API key, model endpoint, and a filename for the image data.
* An HttpURLConnection object is created to connect to the API URL.
* The request method is set to "POST" as the application is sending data to the API.
* Necessary headers are set, including "Content-Type" and "Content-Length" for the encoded image data.
* Output is enabled using connection.setDoOutput(true).
* The encoded image data (byte array) is written to the connection's output stream using an OutputStream.
* The response from the API is then retrieved using a BufferedReader and stored in a StringBuilder object.
* Optionally, the formatJson method might be called to format the JSON response from the API for better readability.
* Finally, the formatted (or raw) response is displayed in the resultTextArea.

**4. Main Method and Execution:**

* The main method is the program's entry point.
* It uses SwingUtilities.invokeLater to ensure proper threading for the Swing UI components.
* Within the run method of the anonymous Runnable, a new instance of InferenceLocal is created and set to be visible, launching the application with the user interface.

# Schematic Diagram



# Source Code

import javax.swing.\*;

import javax.swing.border.EmptyBorder;

import java.awt.\*;

import java.awt.event.\*;

import java.nio.file.Files;

import java.nio.file.Path;

import java.io.\*;

import java.net.\*;

import java.nio.charset.StandardCharsets;

import java.util.Base64;

import javax.imageio.ImageIO;

import java.awt.image.BufferedImage;

public class InferenceLocal extends JFrame {

    private JButton selectImageButton;

    private JTextArea resultTextArea;

    private JLabel imagePreviewLabel;

    public InferenceLocal() {

        setTitle("Image Inference");

        setSize(800, 400); // Increased width to accommodate image preview

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        setLocationRelativeTo(null);

        selectImageButton = new JButton("Select Image");

        selectImageButton.addActionListener(new ActionListener() {

            public void actionPerformed(ActionEvent e) {

                JFileChooser fileChooser = new JFileChooser();

                fileChooser.setDialogTitle("Select Image File");

                int returnValue = fileChooser.showOpenDialog(null);

                if (returnValue == JFileChooser.APPROVE\_OPTION) {

                    File selectedFile = fileChooser.getSelectedFile();

                    processImage(selectedFile.toPath());

                }

            }

        });

        resultTextArea = new JTextArea();

        resultTextArea.setEditable(false);

        resultTextArea.setFont(new Font("Arial", Font.PLAIN, 25)); // Setting font size and style

        JScrollPane scrollPane = new JScrollPane(resultTextArea);

        imagePreviewLabel = new JLabel();

        imagePreviewLabel.setPreferredSize(new Dimension(300, 300)); // Set preferred size for the image preview

        JPanel panel = new JPanel();

        panel.setLayout(new BorderLayout());

        panel.add(selectImageButton, BorderLayout.NORTH);

        panel.add(scrollPane, BorderLayout.CENTER);

        panel.add(imagePreviewLabel, BorderLayout.EAST); // Add image preview to the right

        add(panel);

    }

    private void processImage(Path imagePath) {

        try {

            byte[] imageArray = Files.readAllBytes(imagePath);

            String encoded = Base64.getEncoder().encodeToString(imageArray);

            byte[] data = encoded.getBytes(StandardCharsets.US\_ASCII);

            String api\_key = "KynvI0EJWnGX03nYyGhz"; // Your API Key

            String model\_endpoint = "detected-violence-image-dataset/2"; // Set model endpoint

            // Construct the URL

            String uploadURL = "https://detect.roboflow.com/" + model\_endpoint + "?api\_key=" + api\_key + "&name=original.jpg";

            // Configure Request

            URL url = new URL(uploadURL);

            HttpURLConnection connection = (HttpURLConnection) url.openConnection();

            connection.setRequestMethod("POST");

            connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");

            connection.setRequestProperty("Content-Length", String.valueOf(data.length));

            connection.setDoOutput(true);

            // Write Data

            try (OutputStream os = connection.getOutputStream()) {

                os.write(data);

            }

            // Get Response

            StringBuilder responseContent;

            try (BufferedReader br = new BufferedReader(new InputStreamReader(connection.getInputStream()))) {

                String line;

                responseContent = new StringBuilder();

                while ((line = br.readLine()) != null) {

                    responseContent.append(line);

                }

            }

            // Format JSON output with spacing

            String formattedOutput = formatJson(responseContent.toString());

            resultTextArea.setText(formattedOutput);

            // Set image preview

            BufferedImage image = ImageIO.read(imagePath.toFile());

            Image scaledImage = image.getScaledInstance(300, 300, Image.SCALE\_SMOOTH);

            imagePreviewLabel.setIcon(new ImageIcon(scaledImage));

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

    private String formatJson(String json) {

        // Replace "time" with "time taken to connect to api"

        json = json.replace("\"time\"", "Time Taken to Connect to API ");

        // Replace "predictions" with "Findings"

        json = json.replace("\"predictions\"", "Findings:-");

        // Remove "detection\_id" from each finding

        // Remove "detection\_id" from each finding

        json = json.replaceAll("\"detection\_id\":\"[^\"]\*\",?", "");

        // Rename "class" to "Weapon"

        json = json.replace("\"class\"", "\"Weapon\"");

        // Rename "confidence" to "Violent percentage" and format the value to percentage

        json = json.replaceAll("\"confidence\":(\\d+\\.?\\d\*),", "\"Violent percentage\":$1,");

        // Remove "image" lines

        // Remove "image" lines

        json = json.replaceAll("\"image\":", "IMAGE:-\n");

        // Add one space after "height"

        json = json.replaceAll("\"height\":(\\d+)", "\"height\":$1, ");

        // Add an empty line after "Findings" as a sub-heading

        json = json.replaceAll("\"Findings\":", "\"FINDINGS\":");

        // Remove brackets from the JSON output

        json = json.replaceAll("[\\{\\}\\[\\]]", "");

        // Add line breaks after commas

        json = json.replaceAll(",", ",\n");

        // Remove "x", "y", "width", and "height" from "Findings"

        json = json.replaceAll("\"x\":\\d+\\.?\\d\*,\n\"y\":\\d+\\.?\\d\*,\n\"width\":\\d+\\.?\\d\*,\n\"height\":\\d+\\.?\\d\*,\n", "");

        // Remove '.0' from the "Findings" section

             // Remove '.0,' from the "Findings" section

        json = json.replaceAll("\\.0(?=,\n\"Violent percentage\":)", "");

        // Add indentation

        StringBuilder formattedJson = new StringBuilder();

        String[] lines = json.split("\n");

        int indentLevel = 0;

        for (String line : lines) {

            if (line.endsWith("{")) {

                formattedJson.append(getIndentString(indentLevel)).append(line).append("\n");

                indentLevel*++*;

            } else if (line.endsWith("}")) {

                indentLevel*--*;

                formattedJson.append(getIndentString(indentLevel)).append(line).append("\n");

            } else {

                formattedJson.append(getIndentString(indentLevel)).append(line).append("\n");

            }

        }

        return formattedJson.toString();

    }

    // Method to generate spaces for indentation

    private String getIndentString(int indentLevel) {

        StringBuilder indent = new StringBuilder();

        for (int i = 0; i < indentLevel; i*++*) {

            indent.append("    "); // 4 spaces per indent level

        }

        return indent.toString();

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable() {

            public void run() {

                new InferenceLocal().setVisible(true);

            }

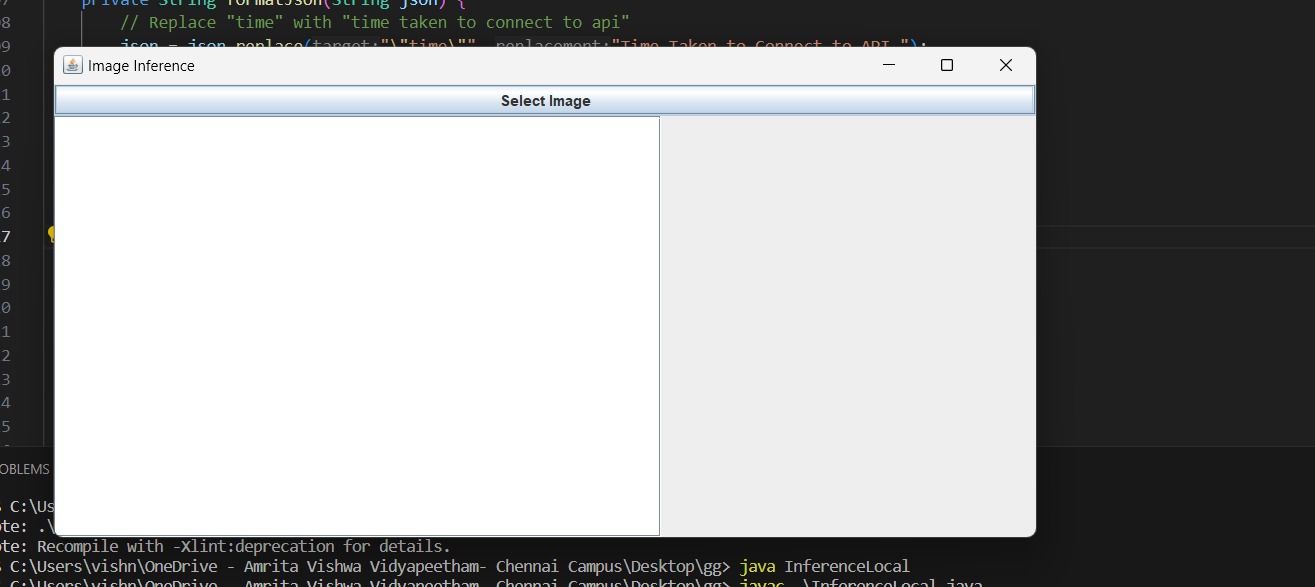
        });

    }

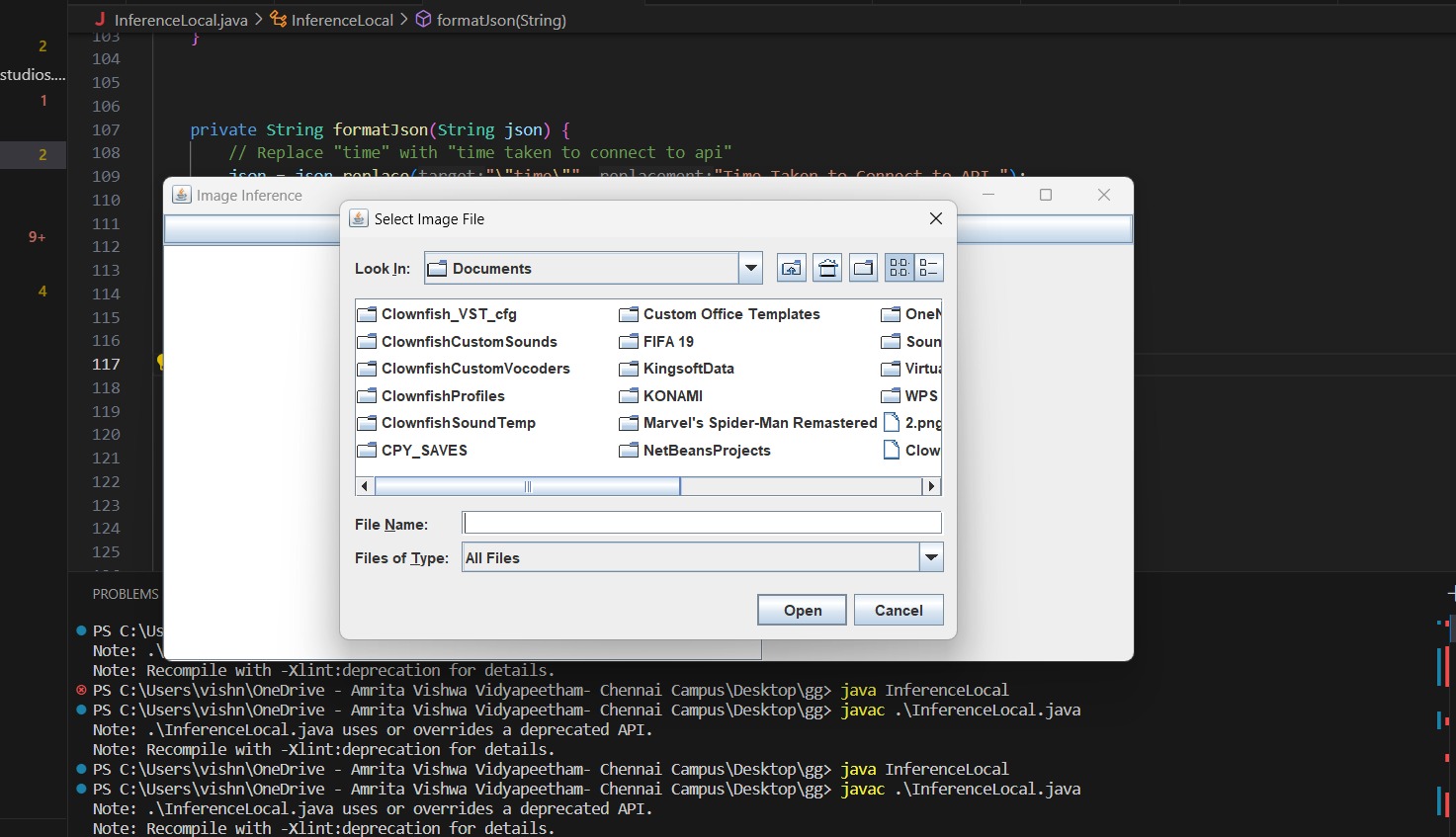
}

# Implementation

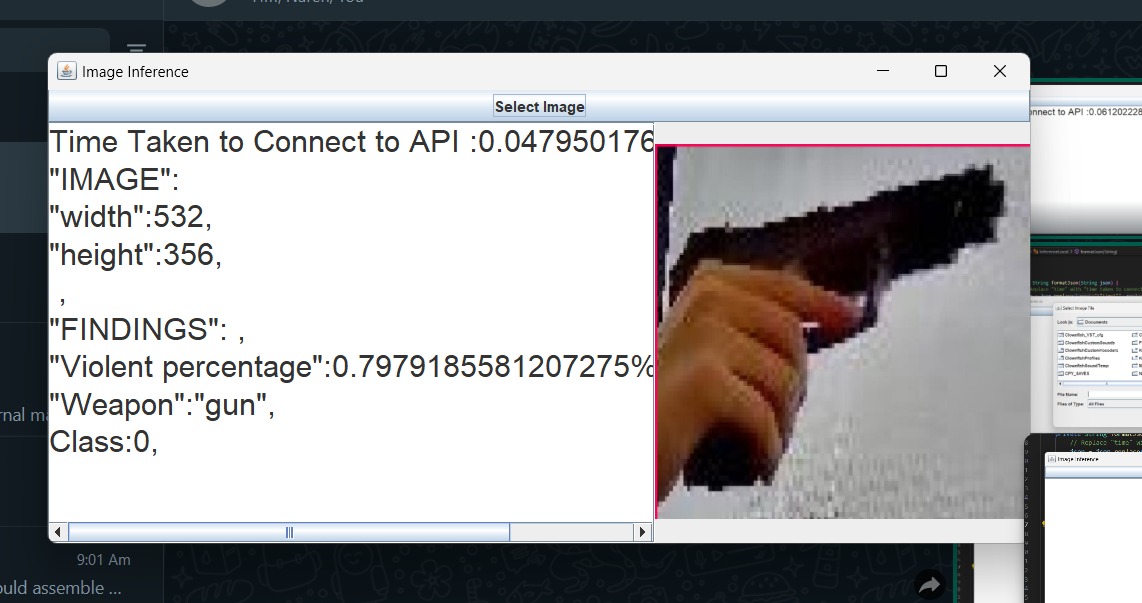
## 1)Select the image :



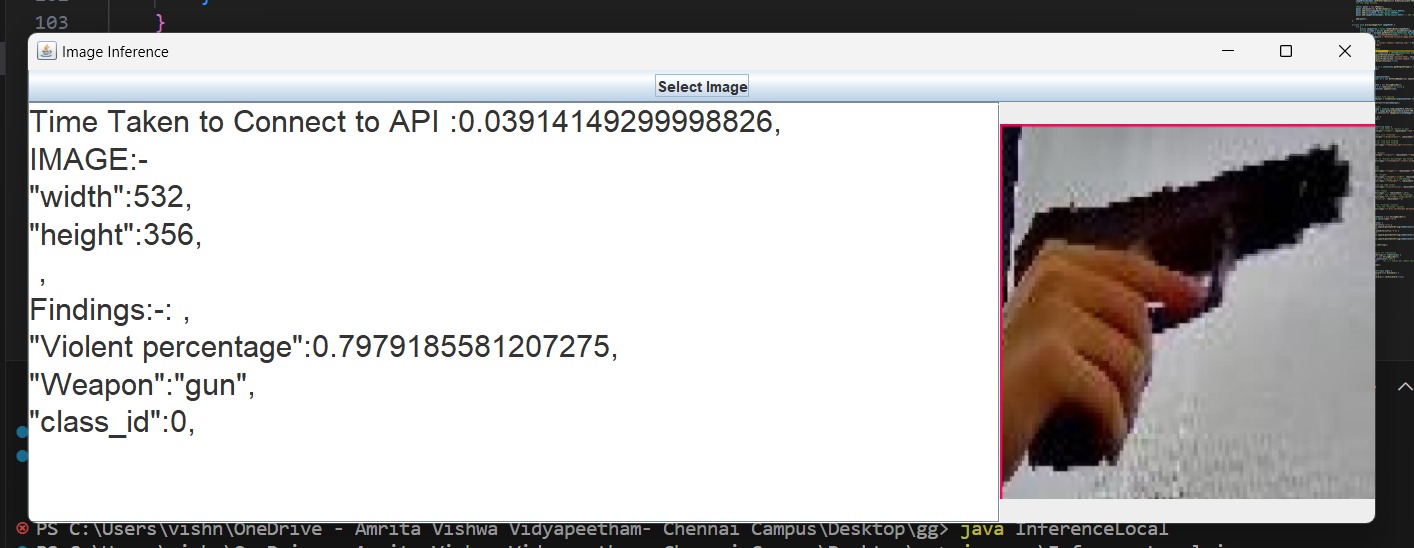
## 2)Mention the path:

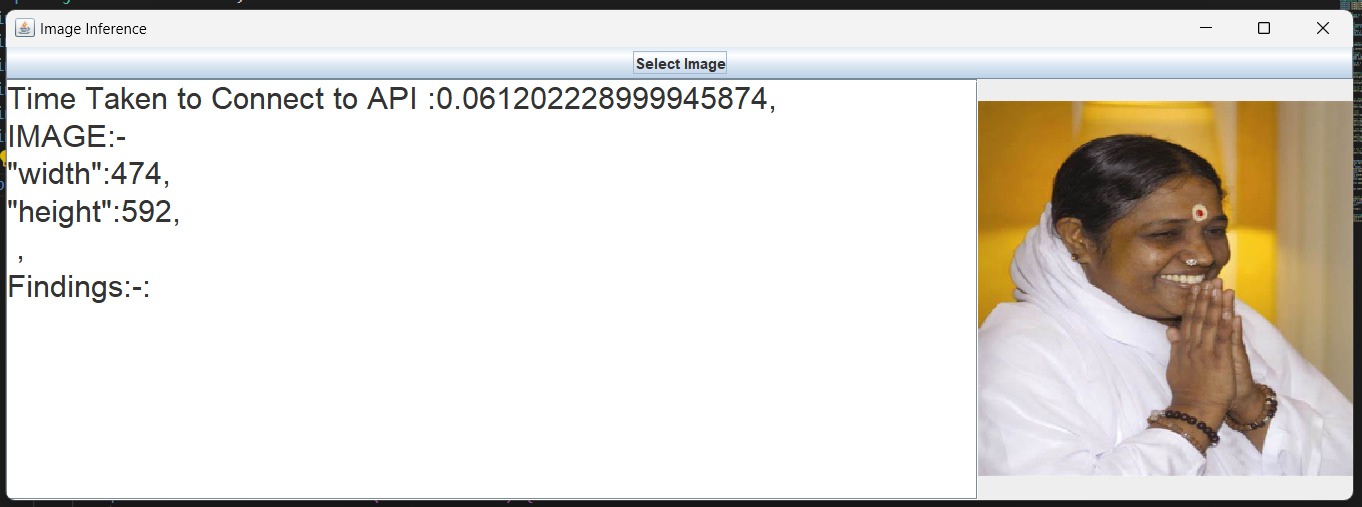


## 3)Wait for the API to communicate among themselves



# Output





# CONCLUSION

1. User Interface Creation (JFrame and Swing Components):

* The code creates a JFrame named "Image Inference" with a set size and default close operation.
* It defines a JButton named "Select Image" for user interaction.
* A JTextArea named resultTextArea is created to display the inference results. This area is non-editable and has a set font style and size.
* A JScrollPane is used to provide scrollable functionality for the results text area within the limited window space.
* A JPanel is created with a BorderLayout layout manager.
* The button and scroll pane are added to the panel at specific locations (North and Center) using the layout manager.
* Finally, the panel is added to the main JFrame window.

2. User Interaction and Image Selection:

* An ActionListener is attached to the "Select Image" button.
* When the button is clicked, a JFileChooser dialog appears, allowing the user to select an image file.
* If the user clicks "Open" (JFileChooser.APPROVE\_OPTION), the selected file's path is retrieved.
* The processImage method is called to handle the selected image.

3. Image Processing and API Interaction:

* The processImage method takes the selected image path as input.
* It reads the entire image file into a byte array using Files.readAllBytes.
* The byte array is then encoded into a Base64 string using Base64.getEncoder().encodeToString.
* The code defines an API key and model endpoint (likely specific to Roboflow).
* It constructs a URL for the inference API call, including the API key, model endpoint, and a filename for the image data.
* An HttpURLConnection object is created to connect to the API URL.
* The request method is set to "POST" as the application is sending data to the API.
* Necessary headers are set, including "Content-Type" and "Content-Length" for the encoded image data.
* Output is enabled using connection.setDoOutput(true).
* The encoded image data (byte array) is written to the connection's output stream using an OutputStream.
* The response from the API is then retrieved using a BufferedReader and stored in a StringBuilder object.
* Optionally, the formatJson method might be called to format the JSON response from the API for better readability.
* Finally, the formatted (or raw) response is displayed in the resultTextArea.

4. Main Method and Execution:

* The main method is the program's entry point.
* It uses SwingUtilities.invokeLater to ensure proper threading for the Swing UI components.
* Within the run method of the anonymous Runnable, a new instance of InferenceLocal is created and set to be visible, launching the application with the user interface.