HAND GESTURE PRESENTATION SYSTEM CODE ANALYSIS

1.Libraries and Imports:

- The code begins by importing necessary libraries and modules. Tkinter (`tk`) is imported for creating the graphical user interface (GUI). Additionally, modules such as `csv`, `datetime`, `cv2` (OpenCV), `os`, and `numpy` are imported for various functionalities like handling CSV files, managing timestamps, working with images and video, interacting with the operating system, and numerical computations.

```
import tkinter as tk
from tkinter import messagebox
import csv
from datetime import datetime
from cvzone.HandTrackingModule import HandDetector
import cv2
import os
import numpy as np
```

2.**Initialization:**

- Two dictionaries, `registered_users` and `gestures`, are initialized to store registered users' credentials and gesture data, respectively.
- Parameters related to the presentation system, such as the dimensions of the camera feed (`width` and `height`), gesture threshold (`gestureThreshold`), and folder path for presentation images (`folderPath`), are set.

```
registered_users = {}
gestures = []
width, height = 1280, 720
gestureThreshold = 300
folderPath = "Presentation"
```

3.**Camera Setup and Hand Detection:**

- The code sets up the camera using OpenCV (`cv2.VideoCapture`) with the specified width and height.
- Initializes the hand detector using the `HandTrackingModule` from `cvzone`. This detector will be used to detect and track hand gestures in the camera feed.

```
cap = cv2.VideoCapture(0)

cap.set(3, width)

cap.set(4, height)

detectorHand = HandDetector(detectionCon=0.8, maxHands=1)
```

4.**Creating the GUI:**

- Tkinter is used to create the main GUI window (`root`). The window is titled "Login and Presentation" and has dimensions of 800x600 pixels.
- A background image (`bg2.png`) is loaded and displayed as the background of the GUI window.
- A heading label is created to display the text "Hand Gesture System Login" with a specified font, size, color, and background.

```
root = tk.Tk()

root.title("Login and Presentation")

root.geometry("800x600")


bg2 = tk.PhotoImage(file="C:/Users/user/PycharmProjects/GUI/bg2.png")

background_label = tk.Label(root, image=bg2)

background_label.place(relwidth=1, relheight=1)


heading_label = tk.Label(root, text="Hand Gesture System Login", font=("Comic Sans
MS", 20, "bold"), fg='white', bg='#223053')

heading_label.pack(pady=50)
```

5.**Functions:**

- `check_credentials_and_run_presentation()`:
    - This function is triggered when the user attempts to log in by clicking the "Login" button.
    - It retrieves the username and password entered by the user from the entry fields.
    - The function reads login credentials from a CSV file (`login_log.csv`), checks if the entered username and password match any records in the file.
    - If the credentials match, it logs the login attempt and displays a message indicating successful login.
    - Finally, it closes the login window and initiates the presentation mode using the `run_presentation()` function, passing the username as a parameter.
    - If the login fails due to invalid credentials, an error message is displayed.

```
def check_credentials_and_run_presentation():

    username = entry_username.get()

    password = entry_password.get()



    with open("login_log.csv", mode="r") as file:

        reader = csv.reader(file)

        for row in reader:

            if row and row[0] == username and row[1] == password:

                log_login(username, password)

                messagebox.showinfo("Login Successful", "Welcome, {}".format(username))

                root.destroy()

                run_presentation(username)

                return



    messagebox.showerror("Login Failed", "Invalid username or password")
```

`run_presentation(username)`:

- This function is called after successful login to initiate the presentation mode.
- It enters a loop where it continuously captures frames from the camera feed, detects hand gestures using the `detectorHand`, and updates the presentation slides accordingly.
- Hand gestures such as moving left (`fingers == [1, 0, 0, 0, 0]`), moving right (`fingers == [0, 0, 0, 0, 1]`), drawing annotations (`fingers == [0, 1, 1, 0, 0]`), and erasing annotations (fingers == [0, 1, 1, 1, 0]) are detected and logged.
- Annotations are drawn on the presentation slides based on the detected hand movements.
- The presentation loop continues until the user exits by pressing the 'q' key.

```
def run_presentation(username):
    global cap, detectorHand, width, height, gestureThreshold, folderPath, cap, imgList, delay, \
        buttonPressed, counter, drawMode, imgNumber, delayCounter, annotations, \
        annotationNumber, annotationStart, hs, ws, pathImages

    while True:
        success, img = cap.read()
        img = cv2.flip(img, 1)
        pathFullImage = os.path.join(folderPath, pathImages[imgNumber])
        imgCurrent = cv2.imread(pathFullImage)

        hands, img = detectorHand.findHands(img)
        cv2.line(img, (0, gestureThreshold), (width, gestureThreshold), (0, 255, 0), 10)
```

```
if hands and buttonPressed is False:

    hand = hands[0]

    cx, cy = hand["center"]

    lmList = hand["lmList"]

    fingers = detectorHand.fingersUp(hand)


    xVal = int(np.interp(lmList[8][0], [width // 2, width], [0, width]))

    yVal = int(np.interp(lmList[8][1], [150, height - 150], [0, height]))

    indexFinger = xVal, yVal


    if cy <= gestureThreshold:

        if fingers == [1, 0, 0, 0, 0]:

            timestamp = datetime.now()

            gesture_name = "Left"

            gestures.append((gesture_name, timestamp))

            log_gestures(username, gesture_name, timestamp)

            buttonPressed = True

            if imgNumber > 0:

                imgNumber -= 1

                annotations = [[]]

                annotationNumber = -1

                annotationStart = False


        if fingers == [0, 0, 0, 0, 1]:
```

```
        timestamp = datetime.now()
        gesture_name = "Right"
        gestures.append((gesture_name, timestamp))
        log_gestures(username, gesture_name, timestamp)
        buttonPressed = True
        if imgNumber < len(pathImages) - 1:
            imgNumber += 1
            annotations = [[]]
            annotationNumber = -1
            annotationStart = False

if fingers == [0, 1, 1, 0, 0]:
    cv2.circle(imgCurrent, indexFinger, 12, (0, 0, 255), cv2.FILLED)
    timestamp = datetime.now()
    gesture_name = "Annotation Pointer"
    gestures.append((gesture_name, timestamp))
    log_gestures(username, gesture_name, timestamp)

if fingers == [0, 1, 0, 0, 0]:
    if annotationStart is False:
        annotationStart = True
        annotationNumber += 1
        annotations.append([])
    annotations[annotationNumber].append(indexFinger)
```

```
        cv2.circle(imgCurrent, indexFinger, 12, (0, 0, 255), cv2.FILLED)
        timestamp = datetime.now()
        gesture_name = "Annotation Painter"
        gestures.append((gesture_name, timestamp))
        log_gestures(username, gesture_name, timestamp)

    else:
        annotationStart = False

    if fingers == [0, 1, 1, 1, 0]:
        if annotations:
            annotations.pop(-1)
            annotationNumber -= 1
            buttonPressed = True
        timestamp = datetime.now()
        gesture_name = "Annotation Eraser"
        gestures.append((gesture_name, timestamp))
        log_gestures(username, gesture_name, timestamp)

else:
    annotationStart = False

if buttonPressed:
    counter += 1
    if counter > delay:
        counter = 0
        buttonPressed = False


for i, annotation in enumerate(annotations):
    for j in range(len(annotation)):
        if j != 0:
            cv2.line(imgCurrent, annotation[j - 1], annotation[j], (0, 0, 200), 12)


imgSmall = cv2.resize(img, (ws, hs))
h, w, _ = imgCurrent.shape
imgCurrent[0:hs, w - ws: w] = imgSmall


cv2.imshow("Slides", imgCurrent)
cv2.imshow("Image", img)
key = cv2.waitKey(1)
if key == ord('q'):
    Break
```

6.Logging Functions:

- `log_gestures(username, gesture_name, timestamp)`:
    - This function logs the gestures made during the presentation to a CSV file specific to each user.
    - It appends each gesture's details (gesture name and timestamp) to the corresponding user's gestures log file.
    - The filename for the log file is constructed using the username.

```
def log_gestures(username, gesture_name, timestamp):

    gestures_filename = f"{username}_gestures_log.csv"



    file_is_empty = not os.path.isfile(f"{username}_gestures_log.csv") or
os.stat(f"{username}_gestures_log.csv").st_size == 0

    with open(gestures_filename, mode='a', newline='') as file:

        writer = csv.writer(file)

        if file_is_empty:

            writer.writerow(["Gesture name", "Timestamp"])

        writer.writerow([gesture_name, timestamp])
```

`log_login(username, password)`:

- This function logs user login attempts along with timestamps to a common login log file (`login_log.csv`).
- It records the username, password, and login timestamp in the CSV file.

```
def log_login(username, password):

  now = datetime.now()

  date_time = now.strftime("%Y-%m-%d %H:%M:%S")



  file_is_empty = not os.path.isfile("login_log.csv") or os.stat("login_log.csv").st_size ==
0



  with open("login_log.csv", mode="a", newline="") as file:

    writer = csv.writer(file)



    if file_is_empty:

      writer.writerow(["Username", "Password", "Timestamp"])



    writer.writerow([username, password, date_time])
```

7.**User Interface Elements:**

- Labels, entry fields, and buttons are created using Tkinter for username, password, login, and registration functionalities.
- Each UI element is packed and placed in the GUI window with appropriate padding and layout.

```
label_username = tk.Label(root, text="Username:", font=("Comic Sans MS", 12),
fg='yellow', bg='#223053')

label_username.pack(pady=10)

entry_username = tk.Entry(root, font=("Helvetica", 14))

entry_username.pack(pady=5)


label_password = tk.Label(root, text="Password:", font=("Comic Sans MS", 12),
fg='yellow', bg='#223053')

label_password.pack(pady=10)

entry_password = tk.Entry(root, show="*", font=("Comic Sans MS", 12))

entry_password.pack(pady=5)


login_button = tk.Button(root, text="Login",
command=check_credentials_and_run_presentation, font=("Comic Sans MS", 12))

login_button.pack(pady=10)


register_button = tk.Button(root, text="Register", command=register_user, font=("Comic
Sans MS", 12))

register_button.pack(pady=20)
```

8.Mainloop and Cleanup

- The `root.mainloop()` function starts the Tkinter main event loop, which handles GUI interactions and events until the application is closed.
- The camera resources are released (`cap.release()`) and OpenCV windows

```
root.mainloop()

# Release the camera when the application is closed

cap.release()

cv2.destroyAllWindows()
```