PYTHON CODE:

from tkinter import messagebox

from PIL import Image, ImageTk

import cv2

import imutils

from scipy.spatial import distance

from imutils import face\_utils

from pygame import mixer

import dlib

# Initialize mixer for sound alert

mixer.init()

mixer.music.load("music.wav")

# Function to compute eye aspect ratio

def eye\_aspect\_ratio(eye):

A = distance.euclidean(eye[1], eye[5])

B = distance.euclidean(eye[2], eye[4])

C = distance.euclidean(eye[0], eye[3])

ear = (A + B) / (2.0 \* C)

return ear

# Global variables for drowsiness detection

thresh = 0.25

frame\_check = 20

detect = dlib.get\_frontal\_face\_detector()

predict = dlib.shape\_predictor(r"C:\Users\rishika\Downloads\shape\_predictor\_68\_face\_landmarks (1).dat")

(lStart, lEnd) = face\_utils.FACIAL\_LANDMARKS\_68\_IDXS["left\_eye"]

(rStart, rEnd) = face\_utils.FACIAL\_LANDMARKS\_68\_IDXS["right\_eye"]

flag = 0

cap = None

monitoring = False

# Dictionary to store registered users (username -> password)

registered\_users = {}

class Application(tk.Tk):

def \_\_init\_\_(self, \*args, \*\*kwargs):

tk.Tk.\_\_init\_\_(self, \*args, \*\*kwargs)

self.geometry("600x400")

self.title("Driver Drowsiness Monitoring System")

container = tk.Frame(self)

container.pack(side="top", fill="both", expand=True)

container.grid\_rowconfigure(0, weight=1)

container.grid\_columnconfigure(0, weight=1)

self.frames = {}

for F in (WelcomePage, LoginPage, RegisterPage, Dashboard, MonitoringPage):

frame = F(container, self)

self.frames[F] = frame

frame.grid(row=0, column=0, sticky="nsew")

self.show\_frame(WelcomePage)

def show\_frame(self, cont):

frame = self.frames[cont]

frame.tkraise()

class WelcomePage(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

label = tk.Label(self, text="Welcome to Driver Drowsiness Monitoring System", font=("Arial", 16))

label.pack(pady=50)

btn\_login = tk.Button(self, text="Proceed to Login", command=lambda: controller.show\_frame(LoginPage))

btn\_login.pack()

class LoginPage(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

label = tk.Label(self, text="Login Page", font=("Arial", 16))

label.pack(pady=20)

self.username = tk.Entry(self)

self.password = tk.Entry(self, show='\*')

self.username.pack(pady=5)

self.password.pack(pady=5)

btn\_login = tk.Button(self, text="Login", command=lambda: self.login(controller))

btn\_login.pack(pady=5)

btn\_register = tk.Button(self, text="Register", command=lambda: controller.show\_frame(RegisterPage))

btn\_register.pack(pady=5)

def login(self, controller):

# Simple login mechanism (add your logic here)

username = self.username.get()

password = self.password.get()

if username in registered\_users and registered\_users[username] == password:

controller.show\_frame(Dashboard)

else:

messagebox.showerror("Login Failed", "Incorrect Username/Password")

class RegisterPage(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

label = tk.Label(self, text="Register Page", font=("Arial", 16))

label.pack(pady=20)

self.new\_username = tk.Entry(self)

self.new\_password = tk.Entry(self, show='\*')

self.new\_username.pack(pady=5)

self.new\_password.pack(pady=5)

btn\_register = tk.Button(self, text="Register", command=self.register)

btn\_register.pack(pady=5)

btn\_back = tk.Button(self, text="Back to Login", command=lambda: controller.show\_frame(LoginPage))

btn\_back.pack(pady=5)

def register(self):

username = self.new\_username.get()

password = self.new\_password.get()

if username and password:

if username not in registered\_users:

registered\_users[username] = password

messagebox.showinfo("Registration Success", "User registered successfully!")

else:

messagebox.showerror("Registration Failed", "Username already exists!")

else:

messagebox.showerror("Registration Failed", "Please enter valid username and password.")

class Dashboard(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

label = tk.Label(self, text="Dashboard", font=("Arial", 16))

label.pack(pady=20)

btn\_home = tk.Button(self, text="Home", command=lambda: controller.show\_frame(Dashboard))

btn\_home.pack(side=tk.LEFT, padx=5)

btn\_about = tk.Button(self, text="About", command=lambda: messagebox.showinfo("About", "This is a Driver Drowsiness Monitoring System."))

btn\_about.pack(side=tk.LEFT, padx=5)

btn\_start\_monitoring = tk.Button(self, text="Start Monitoring", command=lambda: controller.show\_frame(MonitoringPage))

btn\_start\_monitoring.pack(pady=50)

btn\_logout = tk.Button(self, text="Logout", command=lambda: controller.show\_frame(WelcomePage))

btn\_logout.pack(side=tk.BOTTOM, pady=10)

class MonitoringPage(tk.Frame):

def \_\_init\_\_(self, parent, controller):

tk.Frame.\_\_init\_\_(self, parent)

self.controller = controller

self.display\_label = tk.Label(self)

self.display\_label.pack()

btn\_start = tk.Button(self, text="Start Monitoring", command=self.start\_monitoring)

btn\_start.pack(pady=10)

btn\_stop = tk.Button(self, text="Stop Monitoring", command=self.stop\_monitoring)

btn\_stop.pack(pady=10)

def start\_monitoring(self):

global cap, monitoring

if not monitoring:

cap = cv2.VideoCapture(0)

monitoring = True

self.process\_frame()

def stop\_monitoring(self):

global cap, monitoring

monitoring = False

if cap:

cap.release()

self.display\_label.config(image='')

self.controller.show\_frame(Dashboard)

def process\_frame(self):

global flag, monitoring

if not monitoring:

return

ret, frame = cap.read()

if ret:

frame = imutils.resize(frame, width=450)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

subjects = detect(gray, 0)

for subject in subjects:

shape = predict(gray, subject)

shape = face\_utils.shape\_to\_np(shape)

leftEye = shape[lStart:lEnd]

rightEye = shape[rStart:rEnd]

leftEAR = eye\_aspect\_ratio(leftEye)

rightEAR = eye\_aspect\_ratio(rightEye)

ear = (leftEAR + rightEAR) / 2.0

leftEyeHull = cv2.convexHull(leftEye)

rightEyeHull = cv2.convexHull(rightEye)

cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)

cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

if ear < thresh:

flag += 1

if flag >= frame\_check:

cv2.putText(frame, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ALERT!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*", (10, 30),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

cv2.putText(frame, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ALERT!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*", (10, 325),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

if not mixer.music.get\_busy(): # Play sound alert

mixer.music.play()

else:

flag = 0

# Convert the frame to PIL format to display in Tkinter

img = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(img)

imgtk = ImageTk.PhotoImage(image=img)

self.display\_label.imgtk = imgtk

self.display\_label.config(image=imgtk)

# Process the next frame

self.display\_label.after(10, self.process\_frame)

if \_\_name\_\_ == "\_\_main\_\_":

app = Application()

app.mainloop()