|  |
| --- |
| **K. K. Wagh Polytechnic, Nashik.**  HirabaiHaridasVidyanagari, Amrutdham, Panchavati, Nashik-422003  **Department of Computer Technology** |
|  |



**Academic Year: 2023-24**

**Class: TYCM-Win Group No: 26 Date: \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Title of Project: Attendance Monitoring System Using Facial Recognition**

**Assignment No: 2**

**Name of Activity (Assignment): Implemented Code**

**Name: Mrs.Y.U.Kadam**

**Signature of Guide:**

**Main.py:**

from tkinter import \*

from tkinter import ttk, filedialog # Import filedialog module

from tkinter import messagebox

from PIL import Image, ImageTk

import mysql.connector

import csv

import os

from student import Student

from train import Train

from attendance import Attendance

from face\_recog import FaceRecognition

class AnimatedGIFLabel(Label):

def \_\_init\_\_(self, master, filename, width, height, delay=70):

self.image = Image.open(filename)

self.width = width

self.height = height

self.delay = delay

self.frames = [ImageTk.PhotoImage(self.image.resize((width, height)))]

class Face\_Recognition\_System:

def \_\_init\_\_(self, root,login\_window\_destroy\_callback=None):

self.root = root

self.login\_window\_destroy\_callback = login\_window\_destroy\_callback

self.root.geometry("1530x790+0+0")

self.root.title("Student Attendance System")

` def student\_details(self,event):

self.new\_window = Toplevel(self.root)

self.app = Student(self.new\_window)

def train\_data(self,event):

self.new\_window = Toplevel(self.root)

self.app = Train(self.new\_window)

def face\_recog(self,event):

self.new\_window = Toplevel(self.root)

self.app = FaceRecognition(self.new\_window)

def attendance\_stud(self,event):

self.new\_window = Toplevel(self.root)

self.app = Attendance(self.new\_window)

**student.py:**

from tkinter import \*

from tkinter import ttk

from PIL import Image, ImageTk

from tkinter import messagebox

import mysql.connector

import cv2

import os

import re

class Student:

def \_\_init\_\_(self, root):

self.root = root

self.root.geometry("1530x790+0+0")

self.root.title("Student Details")

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Text Variables\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

self.var\_dep = StringVar()

self.var\_course = StringVar()

self.var\_year = StringVar()

self.var\_std\_id = StringVar()

self.var\_std\_name = StringVar()

self.var\_div = StringVar()

self.var\_roll = StringVar()

self.var\_email = StringVar()

self.var\_phone = StringVar()

self.searchCombo = StringVar()

self.searchEntry = StringVar()

def add\_data(self):

if not self.validating\_fields():

return

#validating the empty fields

if self.var\_dep.get()=="Select Department" or self.var\_std\_id.get()=="" or self.var\_std\_name.get()=="":

messagebox.showerror("Error","Fill all the fields!",parent=self.root) #show box on same window

else:

try:

conn = mysql.connector.connect(host="localhost", username="root", password="iamadam4040", database="face\_recognizer")

my\_cursor = conn.cursor()

my\_cursor.execute("INSERT INTO student (Dep, Course, Year, Student\_id, Name, Division, Roll, Email, Phone, PhotoSample) "

"VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)",(

self.var\_dep.get(),

self.var\_course.get(),

self.var\_year.get(),

self.var\_std\_id.get(),

self.var\_std\_name.get(),

self.var\_div.get(),

self.var\_roll.get(),

self.var\_email.get(),

self.var\_phone.get(),

self.var\_radio1.get()

))

conn.commit()

self.fetch\_data() #as soon as save button is clicked, this function is called

conn.close()

messagebox.showinfo("Success","Student details have been added successfully",parent=self.root)

except Exception as es:

messagebox.showerror("Error",f"Due To:{str(es)}",parent=self.root)

def fetch\_data(self):

conn = mysql.connector.connect(host="localhost", username="root", password="iamadam4040", database="face\_recognizer")

my\_cursor = conn.cursor()

my\_cursor.execute("SELECT \* FROM student ORDER BY CAST(Student\_id AS UNSIGNED)")

data = my\_cursor.fetchall()

if len(data)!=0: #if some data is fetched

self.student\_table.delete(\*self.student\_table.get\_children()) #delete the already displayed data

for i in data: #insert new data

self.student\_table.insert("",END,values=i)

conn.commit() #so that data gets added

conn.close()

def delete\_data(self):

if self.var\_std\_id.get()=="": #data will not be deleted if student id is not selected

messagebox.showerror("Error","Student ID required",parent=self.root)

else:

try:

delete = messagebox.askyesno("Delete Data","Are you Sure?",parent=self.root)

if delete>0:

conn = mysql.connector.connect(host="localhost", username="root", password="iamadam4040", database="face\_recognizer")

my\_cursor = conn.cursor()

sql = "delete from student where Student\_id=%s"

val = (self.var\_std\_id.get(),)

my\_cursor.execute(sql,val)

else:

if not delete:

return

conn.commit()

self.fetch\_data()

conn.close()

messagebox.showinfo("Delete","Deleted Successfully",parent=self.root)

except Exception as es:

messagebox.showerror("Error",f"Unsuccessful attempt due to: {str(es)}",parent=self.root)

**train.py:**

from tkinter import \*

from tkinter import ttk

from PIL import Image, ImageTk

from tkinter import messagebox

import face\_recognition

import os

import numpy as np

import json

import xml.etree.ElementTree as ET

class Train:

def \_\_init\_\_(self, root):

self.root = root

self.root.geometry("1530x790+0+0")

self.root.title("Train Data")

def train\_classifier(self):

data\_dir = "Data"

path = [os.path.join(data\_dir, file) for file in os.listdir(data\_dir)]

known\_encodings = []

known\_ids = []

# Load existing encodings and IDs from the XML file

existing\_encodings, existing\_ids = self.load\_existing\_data()

for image\_path in path:

face\_id = os.path.split(image\_path)[1].split('.')[1] # Extract ID from the file name

# Check if data for this ID already exists in the XML file

if face\_id in existing\_ids:

continue # Skip this image if data already exists for this ID

image = face\_recognition.load\_image\_file(image\_path)

face\_encoding = face\_recognition.face\_encodings(image)

if len(face\_encoding) > 0:

known\_encodings.append(face\_encoding[0])

known\_ids.append(face\_id)

# Displaying the training image in a Tkinter window

img = Image.open(image\_path)

img\_tk = ImageTk.PhotoImage(img)

img\_label = Label(self.root, image=img\_tk)

img\_label.image = img\_tk

img\_label.place(x=250, y=150, width=img.width, height=img.height)

self.root.update() # Update the Tkinter window to display the image

self.root.after(10) # Delay for 10 milliseconds

# Append new encodings and IDs to the existing ones and save them in the XML file

self.save\_new\_data(known\_encodings, known\_ids)

messagebox.showinfo("Result", "Data trained successfully!")

def load\_existing\_data(self):

try:

tree = ET.parse("classifier.xml")

root = tree.getroot()

existing\_encodings = [np.array(enc.text.split(","), dtype=np.float64) for enc in root.find("encodings")]

existing\_ids = [id\_elem.text for id\_elem in root.find("ids")]

return existing\_encodings, existing\_ids

except FileNotFoundError:

return [], []

def save\_new\_data(self, new\_encodings, new\_ids):

try:

tree = ET.parse("classifier.xml")

root = tree.getroot()

except FileNotFoundError:

root = ET.Element("root")

encodings\_elem = root.find("encodings")

if encodings\_elem is None:

encodings\_elem = ET.SubElement(root, "encodings")

ids\_elem = root.find("ids")

if ids\_elem is None:

ids\_elem = ET.SubElement(root, "ids")

for encoding in new\_encodings:

encoding\_elem = ET.SubElement(encodings\_elem, "encoding")

encoding\_elem.text = ",".join(map(str, encoding.tolist()))

for face\_id in new\_ids:

id\_elem = ET.SubElement(ids\_elem, "id")

id\_elem.text = str(face\_id)

tree = ET.ElementTree(root)

tree.write("classifier.xml")