**Code Details and Code efficiency:-**

**Admin-site**

**App.py**

from flask import Flask,render\_template,request,redirect,url\_for,session

import MySQLdb

import os

#import for face recongition

from math import sqrt

from sklearn import neighbors

from os import listdir

from os.path import isdir, join, isfile, splitext

import pickle

from PIL import Image, ImageFont, ImageDraw, ImageEnhance

import face\_recognition

from face\_recognition import face\_locations

from face\_recognition.face\_recognition\_cli import image\_files\_in\_folder

app = Flask(\_\_name\_\_)

# getting the location of root directory of the webapp

APP\_ROOT = os.path.dirname(os.path.abspath(\_\_file\_\_))

print(APP\_ROOT)

#connecting Mysql

conn = MySQLdb.connect(host="localhost",user="root",password="",db="login\_info")

#-------------linking navbars------------#

@app.route("/")

def index():

    return render\_template("home.html")

@app.route("/index")

def AdminLoginPage():

    return render\_template("index.html")

@app.after\_request

def set\_response\_headers(response):

    response.headers['Cache-Control'] = 'no-cache, no-store, must-revalidate'

    response.headers['Pragma'] = 'no-cache'

    response.headers['Expires'] = '0'

    return response

@app.route("/signup")

def AdminSignupPage():

    return render\_template("Registration.html")

@app.after\_request

def set\_response\_headers(response):

    response.headers['Cache-Control'] = 'no-cache, no-store, must-revalidate'

    response.headers['Pragma'] = 'no-cache'

    response.headers['Expires'] = '0'

    return response

@app.route("/contact")

def AdminTaskPage():

    return render\_template("contact.html")

@app.route("/Registration")

def BackToHome():

    return render\_template("home.html")

@app.route("/upload")

def Logout():

    return render\_template("home.html")

#---------------#----------------------#

#database

@app.route('/login',methods=['POST'])

def login():

    user = str(request.form["user"])

    paswd = str(request.form["password"])

    cursor = conn.cursor()

    result = cursor.execute("SELECT \* from admin\_login where binary username=%s and binary password=%s",[user,paswd])

    if(result is 1):

        return render\_template("upload.html")

    else:

        return render\_template("index.html",msg="The username or password is incorrect")

@app.route('/signup\_teacher',methods=['POST'])

def signup():

    user = str(request.form["user"])

    paswd = str(request.form["password"])

    email = str(request.form["email"])

    cursor = conn.cursor()

    result = cursor.execute("SELECT \* from teacher\_login where binary username=%s",[user])

    print (result)

    if(result == 1):

        return render\_template("signup.html",title="SignUp",uname=user,msg="already present")

    cursor.execute("INSERT INTO teacher\_login (username,password,email) VALUES(%s, %s, %s)",(user,paswd,email))

    conn.commit()

    return render\_template("Registration.html",msg="successfully signup",uname=user)

@app.route('/signup\_student',methods=['POST'])

def signup\_student():

    user = str(request.form["student\_name"])

    email = str(request.form["student\_email"])

    roll\_id = str(request.form["roll\_id"])

    email1 = str(request.form["parent\_email"])

    cursor = conn.cursor()

    result = cursor.execute("SELECT \* from student\_login where binary username=%s",[user])

    print (result)

    if(result == 1):

        return render\_template("upload.html",uname=user,msg=" already present")

    cursor.execute("INSERT INTO student\_login (username,student\_email,parent\_email,roll\_id) VALUES(%s, %s, %s, %s)",(user,email,email1,roll\_id))

    conn.commit()

    return render\_template("upload.html",uname=user,msg=" successfully signup")

@app.route('/changetask',methods=['POST'])

def changetask():

    return render\_template("home.html")

@app.route("/upload", methods=['POST'])

def upload():

    target = os.path.join(APP\_ROOT,"train\\")

    if not os.path.isdir(target):

        os.mkdir(target)

    classfolder = str(request.form['class\_folder'])

    session['classfolder'] = classfolder

    target1 = os.path.join(target,str(request.form["class\_folder"])+"\\")

    session['target1']=target1

    print(target1)

    model = os.path.join(APP\_ROOT,"model\\")

    if not os.path.isdir(model):

        os.mkdir(model)

    classname = str(request.form['class\_folder']+"\\")

    model = os.path.join(model,classname)

    if not os.path.isdir(model):

        os.mkdir(model)

    session['model']=model

    session['classname'] = classname

    if not os.path.isdir(target1):

        os.mkdir(target1)

    id\_folder = str(request.form["id\_folder"])

    session['id\_folder']= id\_folder

    target2 = os.path.join(target1,id\_folder+"\\")

    if not os.path.isdir(target2):

        os.mkdir(target2)

    target3 = os.path.join(target2,id\_folder+"\\")

    if not os.path.isdir(target3):

        os.mkdir(target3)

    for file in request.files.getlist("file"):

        print(file)

        filename = file.filename

        destination = "\\".join([target3,filename])

        print(destination)

        file.save(destination)

    return call\_train()

def call\_train():

    id\_folder = str(session.get('id\_folder'))

    model=str(session.get('model'))

    if not os.path.isdir(model + id\_folder):

        os.mkdir(model + id\_folder)

    model = model + id\_folder + "\\"

    model = model + "model"

    target1=str(session.get('target1'))

    print(id\_folder)

    print (target1)

    target1 = target1 +id\_folder

    print(target1)

    print(model)

    return train(train\_dir=target1,model\_save\_path=model)

def train(train\_dir, model\_save\_path = "", n\_neighbors = None, knn\_algo = 'ball\_tree', verbose=True):

    id\_folder=str(session.get('id\_folder'))

    X = []

    y = []

    z = 0

    for class\_dir in listdir(train\_dir):

        if not isdir(join(train\_dir, class\_dir)):

            continue

        for img\_path in image\_files\_in\_folder(join(train\_dir, class\_dir)):

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

            faces\_bboxes = face\_locations(image)

            if len(faces\_bboxes) != 1:

                if verbose:

                    print("image {} not fit for training: {}".format(img\_path, "didn't find a face" if len(faces\_bboxes) < 1 else "found more than one face"))

                    os.remove(img\_path)

                    z = z + 1

                continue

            X.append(face\_recognition.face\_encodings(image, known\_face\_locations=faces\_bboxes)[0])

            y.append(class\_dir)

    print(listdir(train\_dir+"\\"+id\_folder))

    train\_dir\_f = listdir(train\_dir+"\\"+id\_folder)

    for i in range(len(train\_dir\_f)):

        if(train\_dir\_f[i].startswith('.')):

            os.remove(train\_dir+"\\"+id\_folder+"\\"+train\_dir\_f[i])

    print(listdir(train\_dir+"\\"+id\_folder))

    if(listdir(train\_dir+"\\"+id\_folder)==[]):

        return render\_template("upload.html",msg1="training data empty, upload again")

    elif(z >= 1):

        return render\_template("upload.html",msg1="Data trained for "+id\_folder+", But one of the image not fit for trainning")

    if n\_neighbors is None:

        n\_neighbors = int(round(sqrt(len(X))))

        if verbose:

            print("Chose n\_neighbors automatically as:", n\_neighbors)

    knn\_clf = neighbors.KNeighborsClassifier(n\_neighbors=n\_neighbors, algorithm=knn\_algo, weights='distance')

    knn\_clf.fit(X, y)

    if model\_save\_path != "":

        with open(model\_save\_path, 'wb') as f:

            pickle.dump(knn\_clf, f)

    return render\_template("upload.html",msg1="Data trained for "+ id\_folder)

if(\_\_name\_\_ == '\_\_main\_\_'):

    app.secret\_key = 'secretkey'

    app.run(host="0.0.0.0",port=4555,debug=True)

**Teachers-site**

**App.py**

from flask import Flask,render\_template,request,redirect,url\_for,session

from flask\_bootstrap import Bootstrap

import MySQLdb

import os

from math import sqrt

from sklearn import neighbors

from os import listdir

from os.path import isdir, join, isfile, splitext

import shutil

import pickle

from PIL import Image, ImageFont, ImageDraw, ImageEnhance

import face\_recognition

from face\_recognition import face\_locations

from face\_recognition.face\_recognition\_cli import image\_files\_in\_folder

from datetime import datetime,timedelta

from pytz import timezone

import xlsxwriter

import pandas as pd

from glob import glob

from flask\_mail import Mail, Message

from io import BytesIO

import base64

import lable\_image

from flask import send\_from\_directory

app = Flask(\_\_name\_\_)

# mail settings

app.config.update(

    DEBUG = True,

    #Email settings

    MAIL\_SERVER = 'smtp.gmail.com',

    MAIL\_PORT = 465,

    MAIL\_USE\_SSL = True,

    MAIL\_USERNAME = 'rameshchauhan00000007@gmail.com',

    MAIL\_PASSWORD = '7715865438',

    MAIL\_DEFAULT\_SENDER = 'rameshchauhan00000007@gmail.com'

    )

mail = Mail(app)

# declaring timezone then creating custom date format

india = timezone('Asia/Kolkata')

date = str(datetime.now(india))[:10] + "@" + str(datetime.now())[11:13] + "hrs"

# getting the location of root directory of the webapp

APP\_ROOT = os.path.dirname(os.path.abspath(\_\_file\_\_))

#APP\_ROOT = os.path.abspath(os.curdir)

print('Path =',APP\_ROOT)

APP\_ROOT1 = APP\_ROOT.split('teachers\_site')

print('Path=',APP\_ROOT1)

# connection with mysql database using python package MySQLdb

conn = MySQLdb.connect(host="localhost",user="root",password="",db="login\_info")

@app.route("/")

def home():

    return render\_template("homepage.html")

@app.after\_request

def set\_response\_headers(response):

    response.headers['Cache-Control'] = 'no-cache, no-store, must-revalidate'

    response.headers['Pragma'] = 'no-cache'

    response.headers['Expires'] = '0'

    return response

@app.route('/login',methods=['POST'])

def login():

    print(APP\_ROOT)

    print(APP\_ROOT1[0])

    user = str(request.form["user"])

    session['user'] = user

    paswd = str(request.form["password"])

    username = user.split(".",1)[0]

    username = str(username)

    print(username)

    print(type(username))

    cursor = conn.cursor()

    result = cursor.execute("SELECT \* from teacher\_login where binary username=%s and binary password=%s",[user,paswd])

    if(result is 1):

        return render\_template("task.html",uname=username)

    else:

        return render\_template("homepage.html",title="Faculty Login",msg="The username or password")

@app.after\_request

def set\_response\_headers(response):

    response.headers['Cache-Control'] = 'no-cache, no-store, must-revalidate'

    response.headers['Pragma'] = 'no-cache'

    response.headers['Expires'] = '0'

    return response

@app.route('/upload\_redirect',methods=['POST'])

def upload\_redirect():

    if(os.path.isfile(APP\_ROOT+"\\image.jpeg")):

        os.remove(APP\_ROOT + "\\image.jpeg")

    return render\_template("upload.html")

@app.route("/upload", methods=['POST'])

def upload():

    if not os.path.isfile(APP\_ROOT+"\\image.jpeg"):

        return render\_template("upload.html",msg="spoof detected")

    id\_folder = str(request.form['id\_folder'])

    session['id\_folder']= id\_folder

    target = os.path.join(APP\_ROOT,"test\\")

    if not os.path.isdir(target):

        os.mkdir(target)

    target1 = os.path.join(target,str(request.form["folder\_name"])+"\\")

    test\_append = str(request.form["folder\_name"])

    session['test\_append']= test\_append

    print(target1)

    if not os.path.isdir(target1):

        os.mkdir(target1)

    shutil.copyfile(APP\_ROOT+"\\"+"image.jpeg",target1+"image.jpeg")

    destination = APP\_ROOT + "\\" + "test\\" + test\_append + "\\" + "image.jpeg"

    session['destination'] = destination

    teacher\_name = str(session.get('user'))

    session['teacher\_name'] = teacher\_name

    #return render\_template("upload.html",msg="uploaded successfully")

    return match()

@app.after\_request

def set\_response\_headers(response):

    response.headers['Cache-Control'] = 'no-cache, no-store, must-revalidate'

    response.headers['Pragma'] = 'no-cache'

    response.headers['Expires'] = '0'

    return response

def match():

    destination = str(session.get('destination'))

    print(destination)

    if os.path.isfile(destination):

        test\_append = str(session.get('test\_append'))

        session['test\_append'] = test\_append

        id\_folder = str(session.get('id\_folder'))

        train\_dir = APP\_ROOT1[0]+"admin\_site\\train\\"+ test\_append

        try:

            model = APP\_ROOT1[0]+"admin\_site\\model\\"+test\_append+"\\" + id\_folder + "\\" +"model"

            print(model)

            return predict1(model)

        except FileNotFoundError:

            os.remove(APP\_ROOT1[0]+"teachers\_site\\image.jpeg")

            return render\_template("upload.html",msg="trained model not present for " + test\_append + ": "+id\_folder)

def predict(X\_img\_path, knn\_clf = None, model\_save\_path ="", DIST\_THRESH = .45):

    if knn\_clf is None and model\_save\_path == "":

        raise Exception("must supply knn classifier either thourgh knn\_clf or model\_save\_path")

    if knn\_clf is None:

        with open(model\_save\_path, 'rb') as f:

            knn\_clf = pickle.load(f)

    X\_img = face\_recognition.load\_image\_file(X\_img\_path)

    X\_faces\_loc = face\_locations(X\_img)

    if len(X\_faces\_loc) == 0:

        return []

    faces\_encodings = face\_recognition.face\_encodings(X\_img, known\_face\_locations=X\_faces\_loc)

    closest\_distances = knn\_clf.kneighbors(faces\_encodings, n\_neighbors=1)

    is\_recognized = [closest\_distances[0][i][0] <= DIST\_THRESH for i in range(len(X\_faces\_loc))]

    return [(pred) if rec else ("unknown") for pred, rec in zip(knn\_clf.predict(faces\_encodings), is\_recognized)]

def predict1(model):

    test\_append = str(session.get('test\_append'))

    test\_dir = APP\_ROOT1[0]+"teachers\_site\\test\\" + test\_append

    f\_preds = []

    for img\_path in listdir(test\_dir):

        preds = predict(join(test\_dir, img\_path) ,model\_save\_path=model)

        f\_preds.append(preds)

        print(f\_preds)

    print(len(preds))

    print(len(f\_preds))

    for i in range(len(f\_preds)):

        if(f\_preds[i]==[]):

            os.remove(APP\_ROOT1[0]+"teachers\_site\\image.jpeg")

            return render\_template("upload.html",msg="upload again, face not found")

        else:

            os.remove(APP\_ROOT1[0]+"teachers\_site\\image.jpeg")

    excel = os.path.join(APP\_ROOT,"excel\\")

    if not os.path.isdir(excel):

        os.mkdir(excel)

    excel1 = os.path.join(excel,test\_append)

    if not os.path.isdir(excel1):

        os.mkdir(excel1)

    teacher\_name = str(session.get('teacher\_name'))

    excel2 = os.path.join(excel1,teacher\_name)

    if not os.path.isdir(excel2):

        os.mkdir(excel2)

    session['excel2'] = excel2

    excel3 = excel2+"\\"+date+'.xlsx'

    if not os.path.isfile(excel3):

        workbook = xlsxwriter.Workbook(excel2+"\\"+date+'.xlsx')

        worksheet = workbook.add\_worksheet()

        worksheet.set\_column(0,0,20)

        worksheet.write('A1','Roll Id')

        f\_preds.sort()

        row = 1

        col = 0

        for i in range(len(f\_preds)):

            for j in range(len(f\_preds[i])):

                worksheet.write\_string(row,col,f\_preds[i][j])

                row += 1

        workbook.close()

        return render\_template("upload.html",msg= f\_preds[0][0] + " present")

    else:

        df = pd.read\_excel(excel2+"\\"+date+'.xlsx')

        writer = pd.ExcelWriter(excel2 + "\\" + date+'.xlsx')

        df.to\_excel(writer,sheet\_name="Sheet1",index=False)

        workbook  = writer.book

        worksheet = writer.sheets['Sheet1']

        rows=df.shape[0]

        worksheet.write\_string(rows+1,0,f\_preds[0][0])

        writer.save()

        df = pd.read\_excel(excel2+"\\"+date+'.xlsx')

        df.drop\_duplicates(['Roll Id'],keep='first',inplace=True)

        result = df.sort\_values("Roll Id")

        writer = pd.ExcelWriter(excel2 + "\\" + date+'.xlsx')

        result.to\_excel(writer,'Sheet1',index=False)

        workbook = writer.book

        worksheet = writer.sheets['Sheet1']

        worksheet.set\_column(0,0,20)

        writer.save()

        return render\_template("upload.html",msg= f\_preds[0][0] + " present")

@app.route('/view\_report',methods=['POST'])

def view\_report():

    return render\_template("excel.html")

# view route to download excel files

@app.route('/view',methods=['POST'])

def view():

    test\_append = str(request.form['folder\_name'])

    session['test\_append']=test\_append

    teacher\_name = str(session.get('user'))

    excel\_dir = APP\_ROOT+"\\excel\\"+test\_append+"\\"+teacher\_name+"\\"

    excel\_date = request.form['fname']

    time = request.form['ftime']

    time = time[:2]

    print(time)

    final\_excel=glob(excel\_dir + "\\" + excel\_date+ "@" + time +"\*.xlsx")[0]

    print(final\_excel)

    df = pd.read\_excel(final\_excel)

    df.index += 1

    return render\_template("files.html",msg=final\_excel,df=df,date=excel\_date+"@"+time+"hrs")

@app.route('/excel/<path:filename>', methods=['POST'])

def download(filename):

    return send\_from\_directory(directory='excel', filename=filename)

# route to send emails to parents and students

@app.route('/send\_mail',methods=['POST'])

def send\_mail():

    test\_append = str(request.form['folder\_name'])

    teacher\_name = str(session.get('user'))

    excel\_dir = APP\_ROOT+"\\excel\\"+test\_append+"\\"+teacher\_name+"\\"

    excel\_date = request.form['fname']

    time = request.form['ftime']

    time = time[:2]

    final\_send = glob(excel\_dir + "\\" + excel\_date+ "@" + time +"\*.xlsx")[0]

    print(final\_send)

    df = pd.read\_excel(final\_send)

    roll\_id = list(df['Roll Id'])

    print(type(roll\_id))

    print(roll\_id)

    cursor = conn.cursor()

    for i in range(len(roll\_id)):

        cursor.execute("SELECT student\_email,parent\_email from student\_login where binary roll\_id=%s",[roll\_id[i]])

        email = list(cursor.fetchone())

        print(type(email[1]))

        print(email[0])

        print(email[1])

        msg = Message('Auto Generated',recipients= [email[0],email[1]])

        msg.body = "Hi.. " + roll\_id[i] + " is present for the lecture of " + "Prof. " +str(teacher\_name.split('.',1)[0]) + ", which is held on " + excel\_date + "@" + time + "hrs"

        msg.html = "Hi.. " + roll\_id[i] + " is present for the lecture of " + "Prof. " +str(teacher\_name .split('.',1)[0])+ ", which is held on " + excel\_date + "@" + time + "hrs"

        mail.send(msg)

    return "<h1>mail sent<h1>"

#.........update attendance .......#

@app.route('/update',methods=['POST'])

def update():

    test\_append = str(request.form['excel\_folder'])

    print(test\_append)

    teacher\_name = str(session.get('user'))

    print(teacher\_name)

    excel\_dir = APP\_ROOT + "\\excel\\" + test\_append + "\\" + teacher\_name + "\\"

    print(excel\_dir)

    for file in request.files.getlist("updated\_excel"):

        print(file)

        filename = file.filename

        print(filename)

        destination = "\\".join([excel\_dir,filename])

        print(destination)

        file.save(destination)

    return render\_template("excel.html",msg="updated successfully")

@app.route('/calculate',methods=['POST'])

def calculate():

    test\_append = str(request.form['final\_class'])

    print(test\_append)

    teacher\_name = str(session.get('user'))

    print(teacher\_name)

    excel\_root = APP\_ROOT + "\\excel\\" + test\_append + "\\" + teacher\_name + "\\"

    print(excel\_root)

    excel\_names = os.listdir(excel\_root)

    print(excel\_names)

    for i in range(len(excel\_names)):

        if excel\_names[i].startswith("."):

            os.remove(excel\_root+excel\_names[i])

        else:

            if os.path.isdir(excel\_root+excel\_names[i]):

                shutil.rmtree(excel\_root+excel\_names[i], ignore\_errors=False, onerror=None)

    excel\_names = os.listdir(excel\_root)

    if(excel\_names==[]):

        return render\_template("excel.html",msg1="No excel files found")

    for i in range(len(excel\_names)):

        excel\_names[i] = excel\_root + excel\_names[i]

    print(type(excel\_names))

    # read them in

    excels = [pd.ExcelFile(name) for name in excel\_names]

    # turn them into dataframes

    frames = [x.parse(x.sheet\_names[0], header=None,index\_col=None) for x in excels]

    # delete the first row for all frames except the first

    # i.e. remove the header row -- assumes it's the first

    frames[1:] = [df[1:] for df in frames[1:]]

    # concatenate them..

    combined = pd.concat(frames)

    if not os.path.isdir(excel\_root+"final\\"):

        os.mkdir(excel\_root + "final\\")

    final = excel\_root + "final\\"

    print(final)

    # write it out

    combined.to\_excel(final+"final.xlsx", header=False, index=False)

    # below code is to find actual repetative blocks

    workbook = pd.ExcelFile(final+"final.xlsx")

    df = workbook.parse('Sheet1')

    sample\_data = df['Roll Id'].tolist()

    print (sample\_data)

    #a dict that will store the poll results

    results = {}

    for response in sample\_data:

        results[response] = results.setdefault(response, 0) + 1

    finaldf = (pd.DataFrame(list(results.items()), columns=['Roll Id', 'Total presenty']))

    finaldf = finaldf.sort\_values("Roll Id")

    print (finaldf)

    writer = pd.ExcelWriter(final+"final.xlsx")

    finaldf.to\_excel(writer,'Sheet1',index=False)

    workbook  = writer.book

    worksheet = writer.sheets['Sheet1']

    worksheet.set\_column(0,1,20)

    writer.save()

    final = final + "final.xlsx"

    session['final']=final

    final = final[91:]

    return viewfinal(final)

def viewfinal(final):

    test\_append = str(session.get('test\_append'))

    final\_path = str(session.get('final'))

    df = pd.read\_excel(final\_path)

    df.index += 1

    return render\_template("files.html",msg=final,course=test\_append,df=df)

@app.route('/changetask',methods=['POST'])

def changetask():

    return render\_template("task.html")

@app.route('/logout',methods=['POST'])

def logout():

    return render\_template("index.html",title="Faculty Login",msg1="Logged out please login again")

@app.route('/hello',methods=['POST'])

def hello():

    data\_url = request.values['imageBase64']

    data\_url= data\_url[22:]

    im = Image.open(BytesIO(base64.b64decode(data\_url)))

    print(type(im))

    im.save('image.jpeg')

    filepath = APP\_ROOT + "\\" + "image.jpeg"

    var = lable\_image.function(filepath)

    print(var)

    for i in range(len(var)):

        if(var[i] > 0.8):

            os.remove(filepath)

    return ''

if(\_\_name\_\_ == '\_\_main\_\_'):

    app.secret\_key = 'super secret key'

    app.run(host="0.0.0.0",port=4555,debug=True)

**Label-image.py**

# This requires retrained\_labels.txt and retrained\_graph.pb files which are available at this link.

# https://drive.google.com/open?id=1TMCTLEqekAh1zMwdSfWFxMKNKYyfgIYB

# https://drive.google.com/open?id=1d93B2ehbxrpbAhm7OqtvgDT5Mszk4\_jc

# Change both file path accordingly.

import tensorflow as tf,sys

import os

APP\_ROOT = os.path.dirname(os.path.abspath(\_\_file\_\_))

print('path=',APP\_ROOT)

def function(xyz):

    output = []

    image\_path = xyz

    image\_data = tf.io.gfile.GFile(image\_path,'rb').read()

    label\_lines = [line.rstrip() for line

                            in tf.io.gfile.GFile(APP\_ROOT +"\\tensor\\tf\_files\\retrained\_labels.txt")]

    with tf.io.gfile.GFile(APP\_ROOT +"\\tensor\\tf\_files\\retrained\_graph.pb",'rb') as f:

        graph\_def = tf.compat.v1.GraphDef()

        graph\_def.ParseFromString(f.read())

        \_ = tf.import\_graph\_def(graph\_def,name='')

    with tf.compat.v1.Session() as sess:

        softmax\_tensor = sess.graph.get\_tensor\_by\_name('final\_result:0')

        predictions = sess.run(softmax\_tensor, \

            {'DecodeJpeg/contents:0': image\_data})

        top\_k = predictions[0].argsort()[-len(predictions[0]):][::-1]

        i = 0

        for node\_id in top\_k:

            human\_string = label\_lines[node\_id]

            score = predictions[0][node\_id]

            print('%s (score = %.5f)' % (human\_string,score))

            output.append(score)

    return output

5.4 Modifications and Improvements

After testing and finding all errors, exceptions and bugs, errors were solved by troubleshooting, exceptions were handled by try catch block at the correct places avoiding redundancy of code from model class. Bugs can never be removed completely but most of them are solved and remaining are under development.

# 6. RESULTS AND DISCUSSION

**6.1. User Documentation:-**

There are mainly two webapps for this project one is say admin site and other one is teacher's site.

The whole concept is at the time of admission to college or school admin should register the students details such as his name email address and also create training data of each student by entering his roll id and taking snaps of his or her frontal face and then webapp will automatically create model for that particular roll id and save it on server, The model which is created for each student is about 8kb in size. Admin can also register teachers using this site.

Now using teacher's site (It will be used when teacher will actually enter the class), teacher has to login first and then after clicking on attendance tab there will be no back button as teacher will pass on the phone to student.

Student's will then have to just click a snap enter class and roll id and press enter to mark their attandance.

After that there is also a problem of spoof attack in face recognition i.e. someone will show someone's image of face through their mobile phone and trick our webapp and they will mark the attendace of their friends who were not present.

But I solved this issue using tensorflow, by training inception to detect mobile phones in an image then I used that model in the webapp as soon as student's click a snap it will first check if the face is spoof or original. To retrain inception's last layer I used 200 images of mobile phones and I feed them to tensorflow to retrain the last layer of inception. To do this follow <https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0> To download 200 images at once use fatkun-batch-download chrome extension.

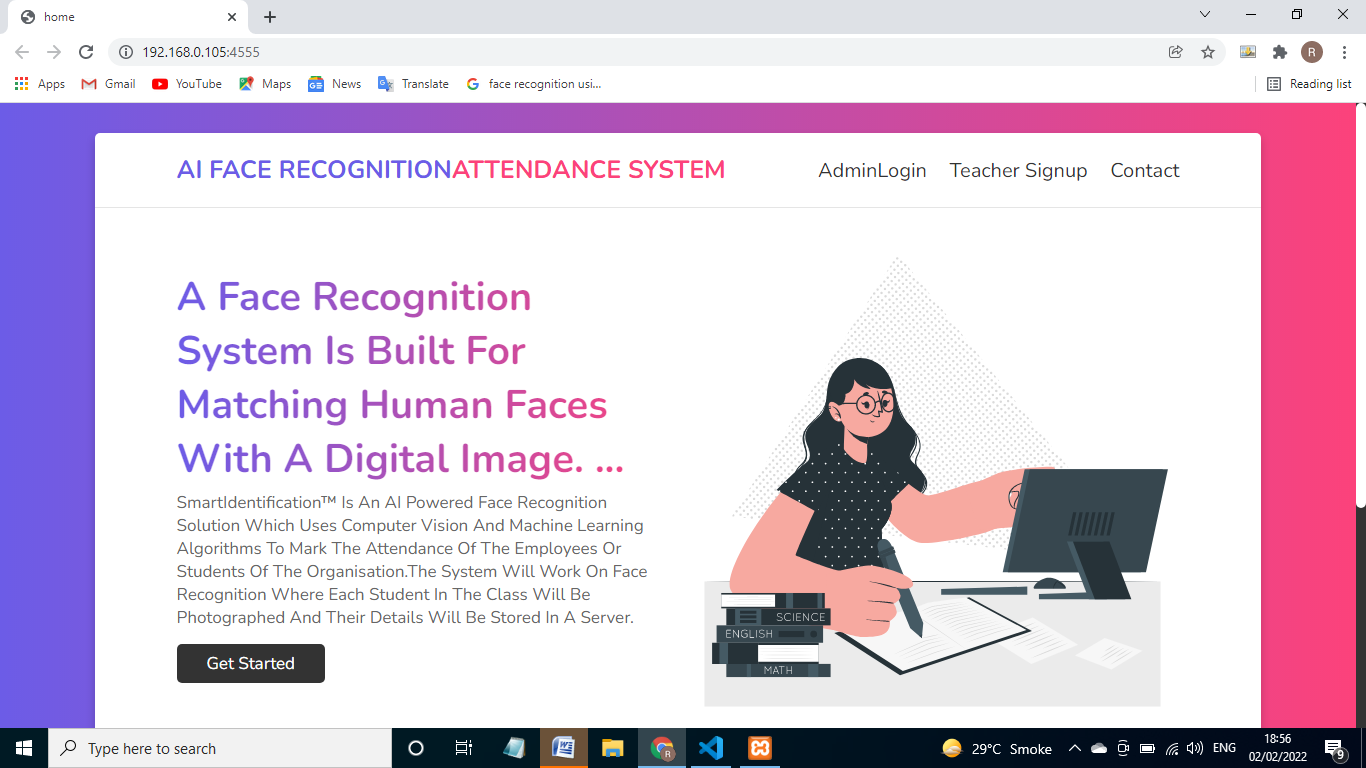
Now after student's part is done. Teacher can then login again and then go to report's tab to see attendance. Here there are several option's I have given.

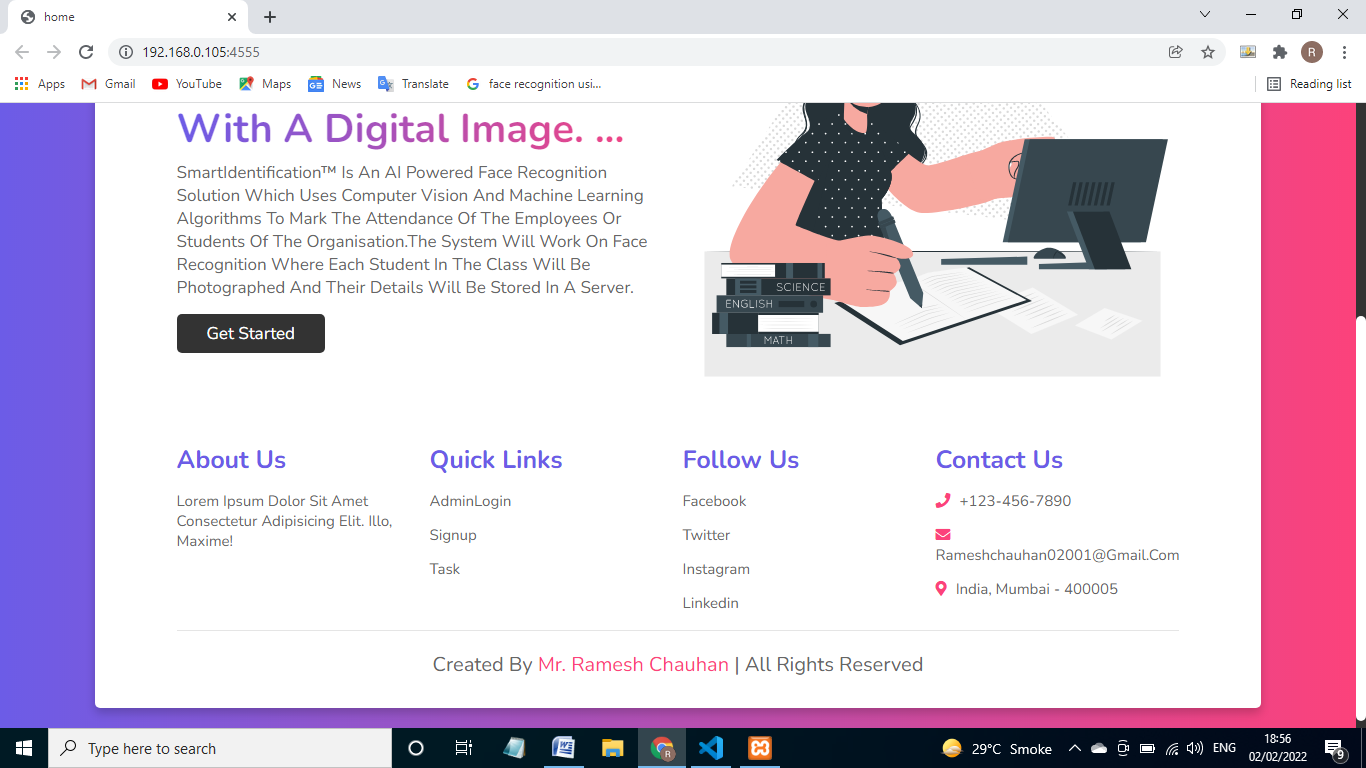
If teacher want to see today's attendace, just select date and time to see the attendance. And there is also an option to download the attendace sheet in excel form and then again reupload it after making any changes if sometime required by the teacher. And the teacher can also see total attendance for his or her lecture. so that they can analyze how many lectures each student from particular class had attended so far.

One additional feature is that teacher can send email for the attendance marked to all the parents as well as students by selecting class and clicking on send mail button.

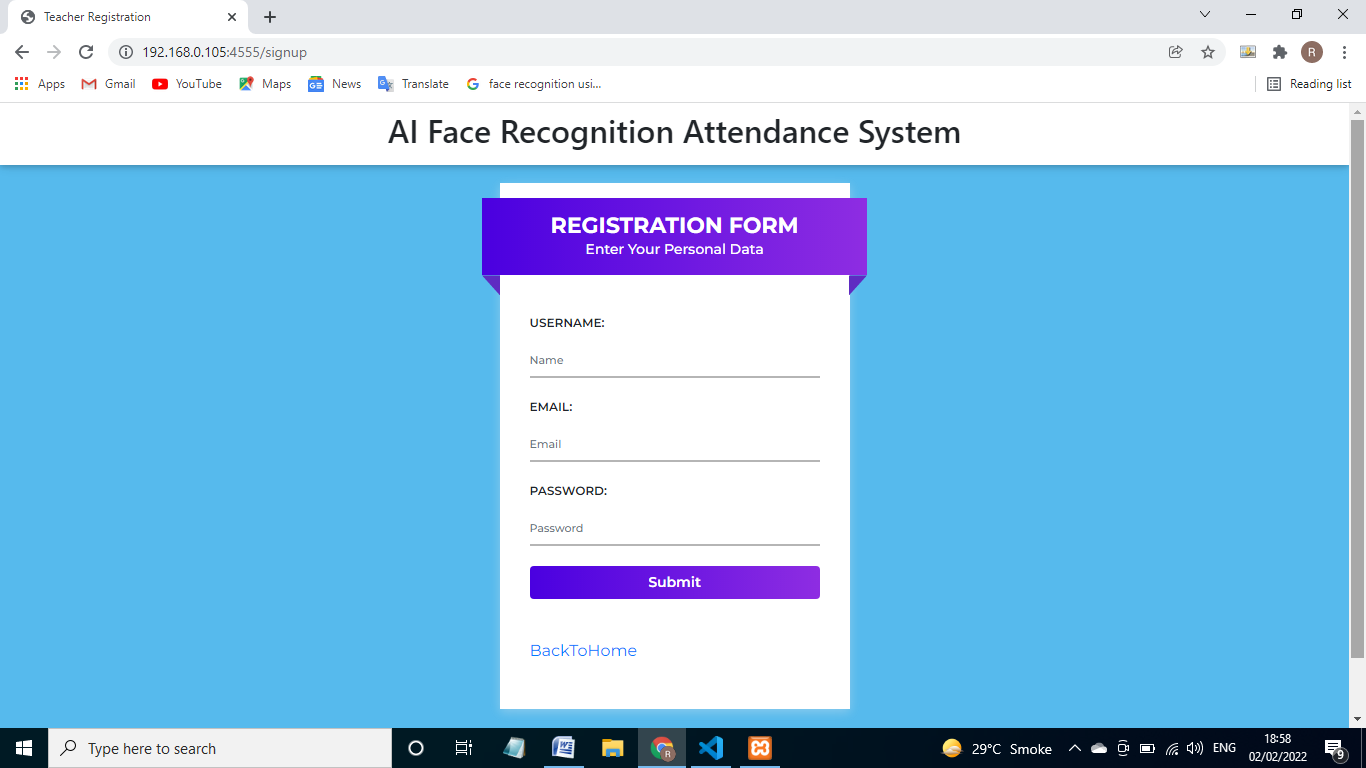
**Admin Site**

**Homepage**

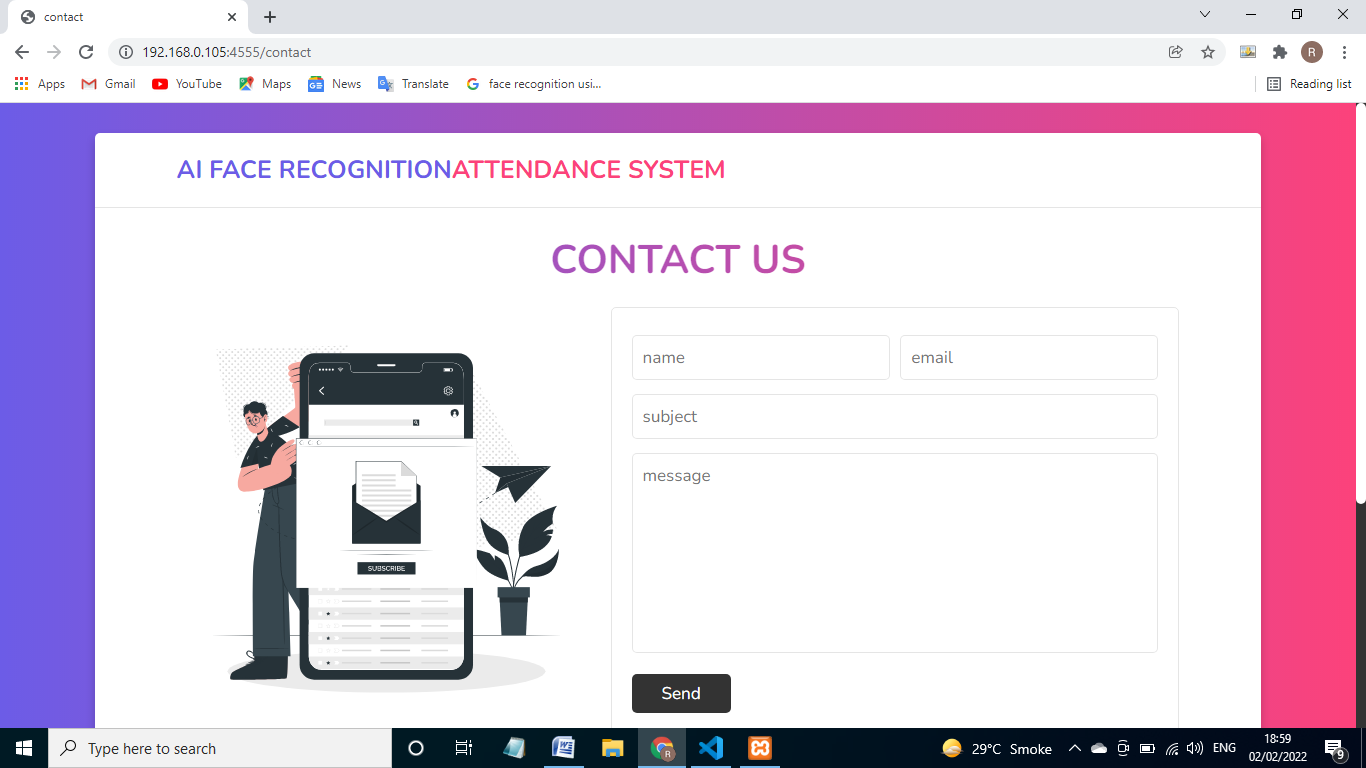
****

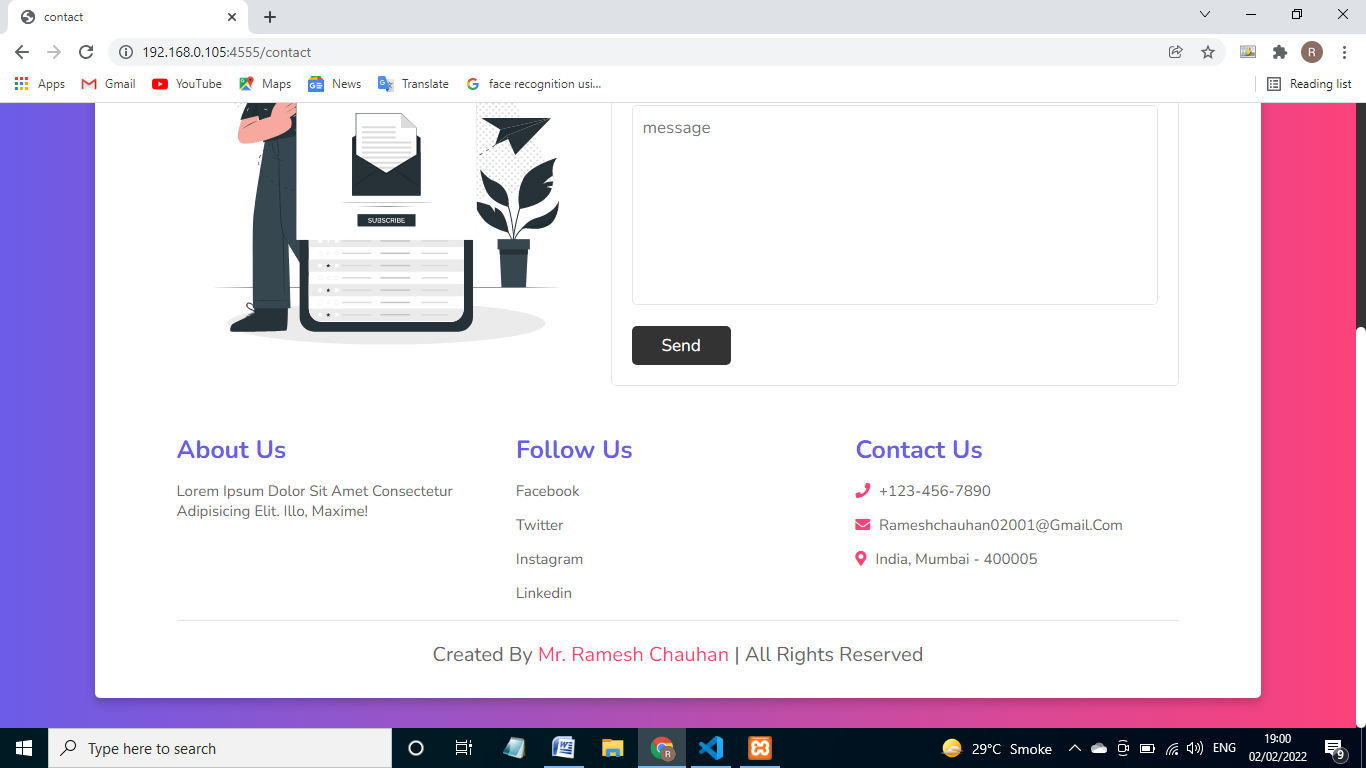
****

**Teacher Registration page:-**

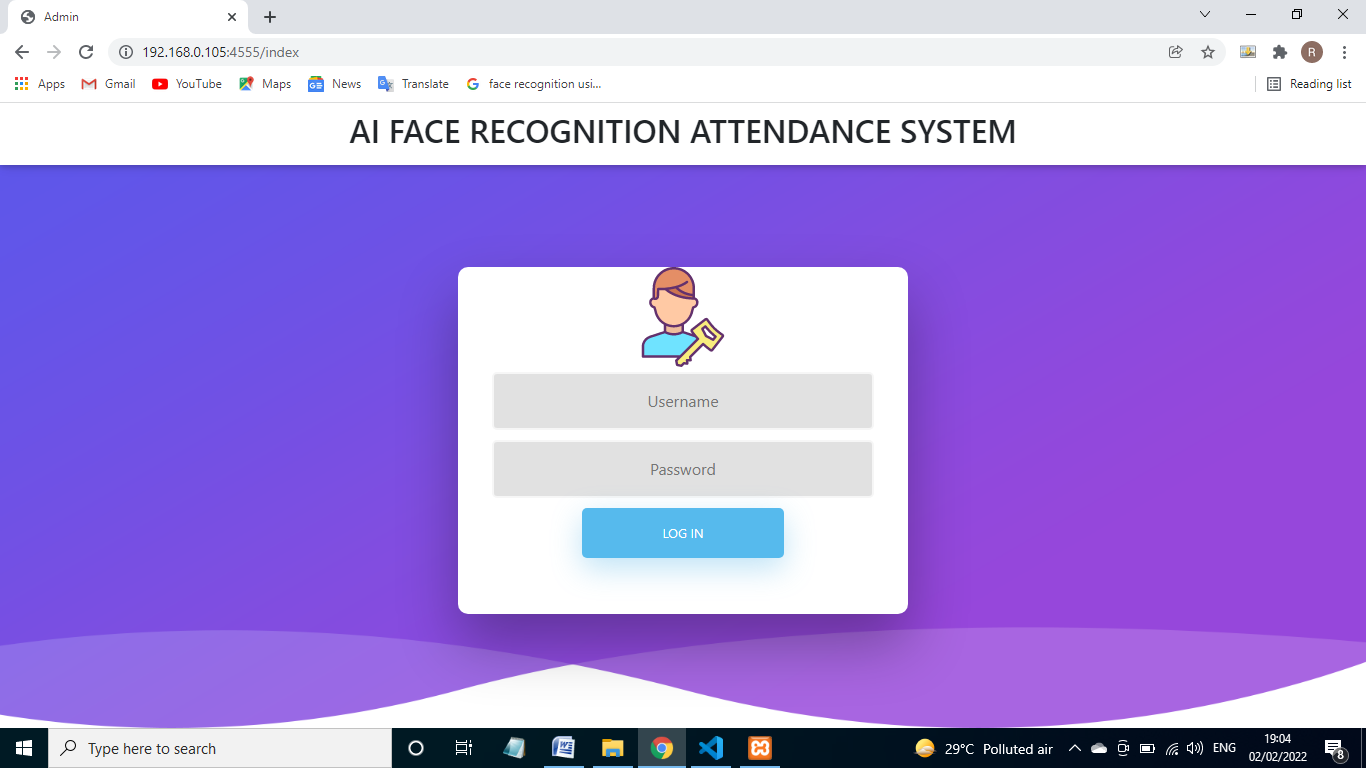
****

**Contact Page:-**

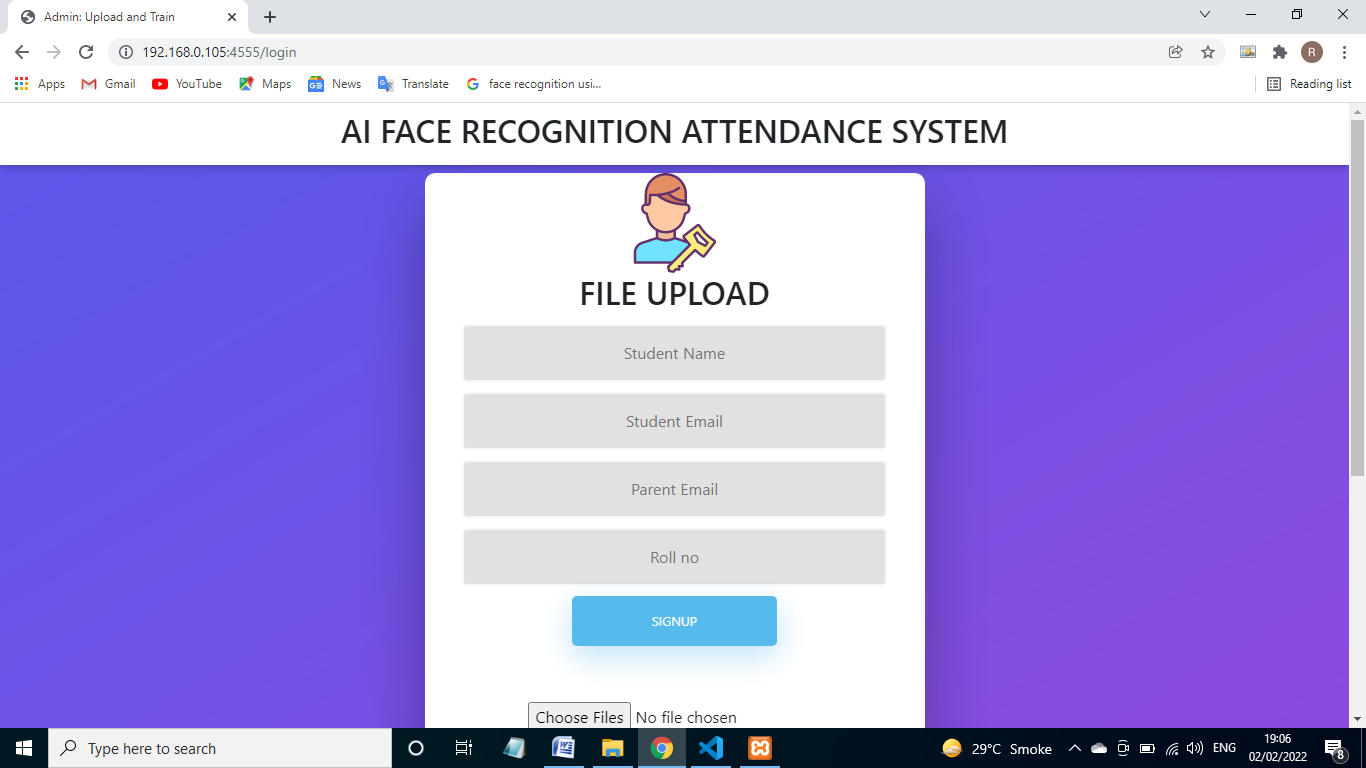
****

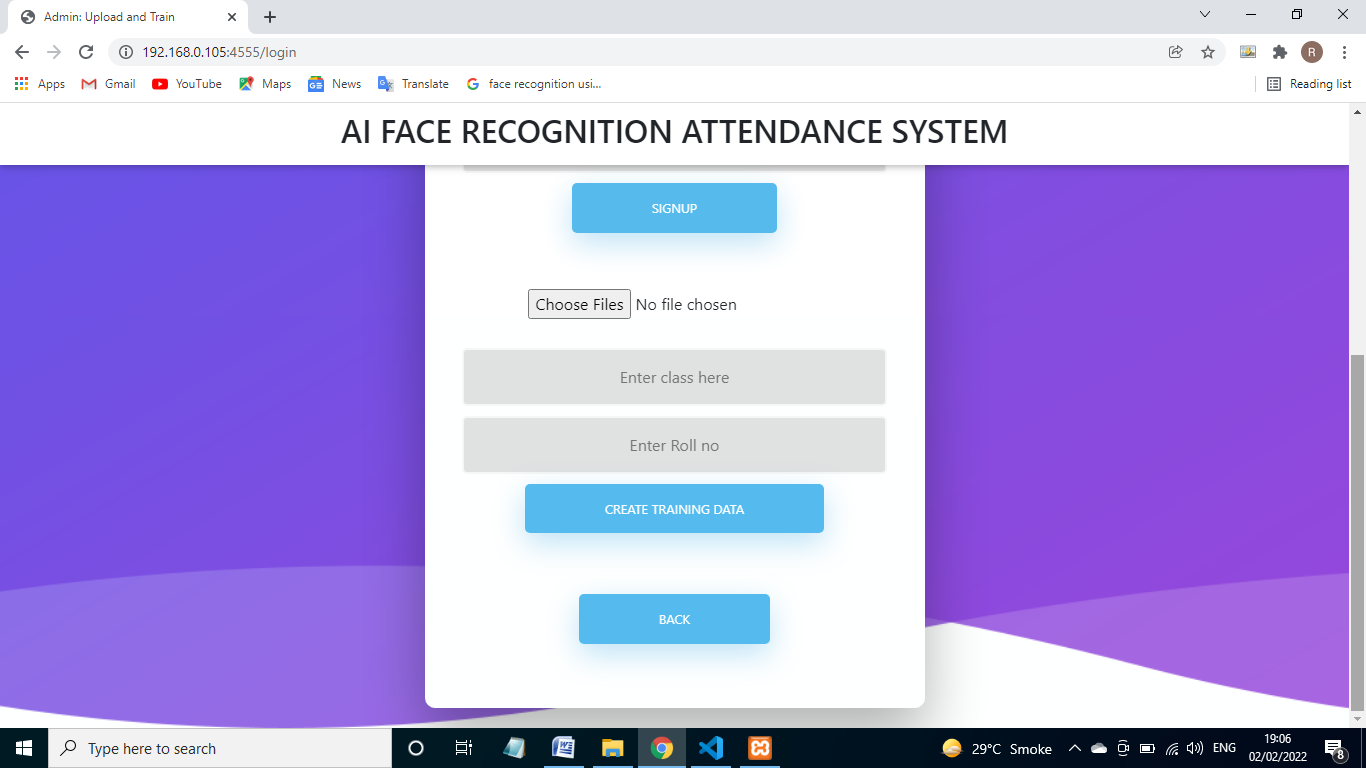
****

**Admin login Page**

****

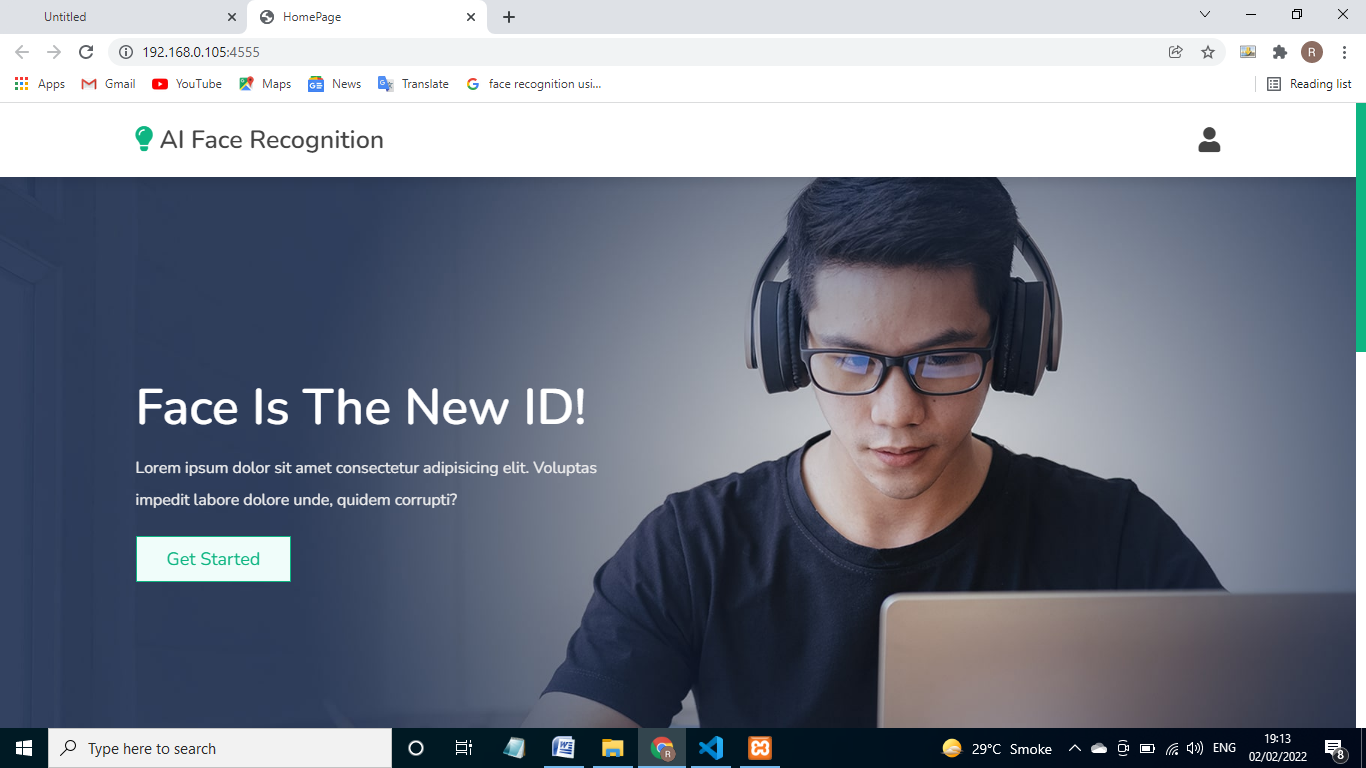
**Student Registration and Model Training Page**

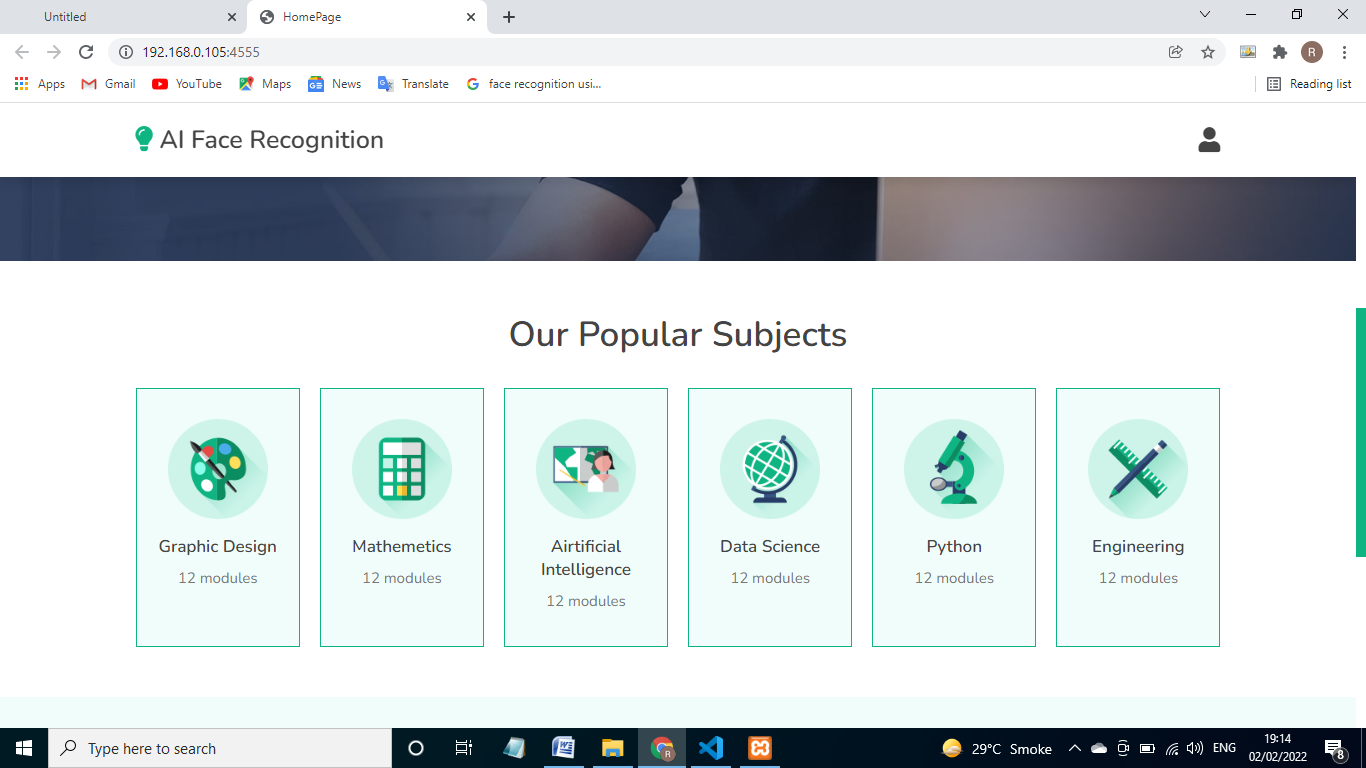
****

****

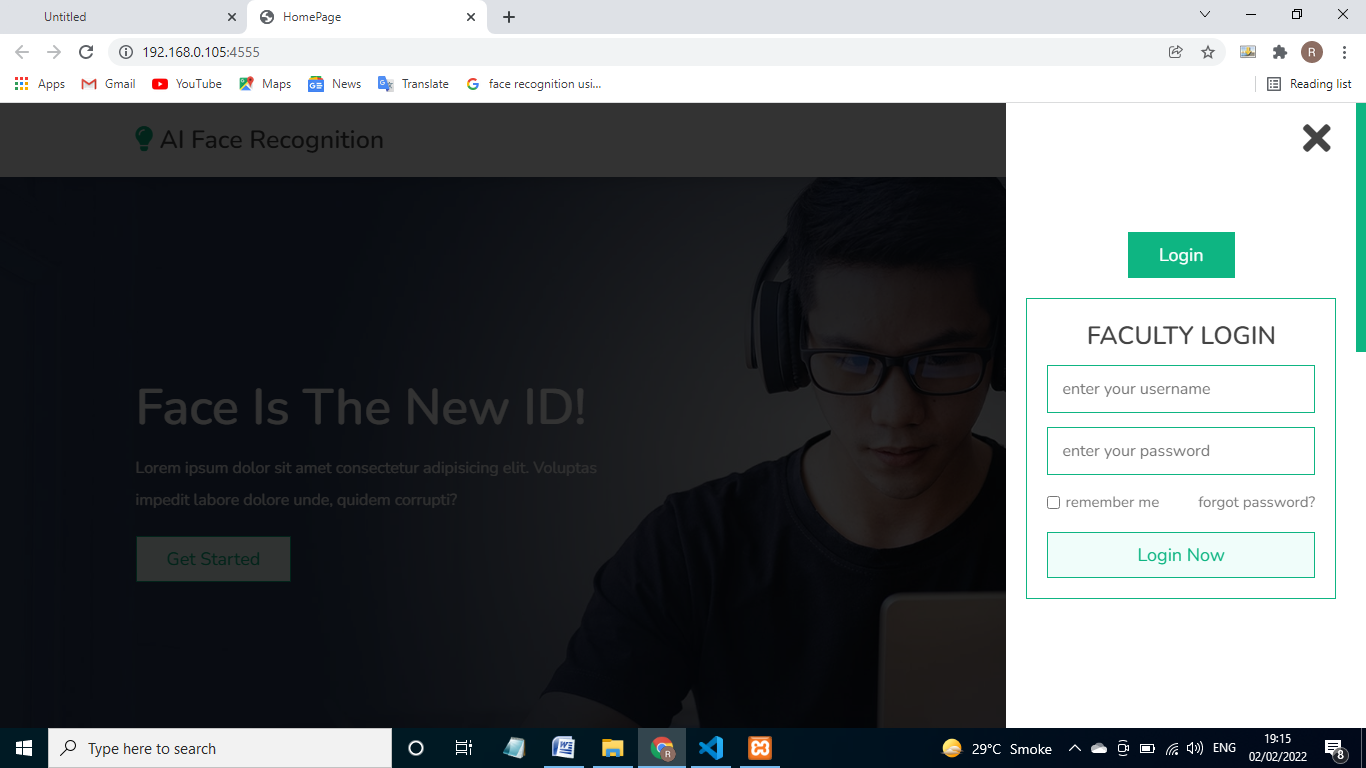
**Teachers Site:**

**Homepage:**

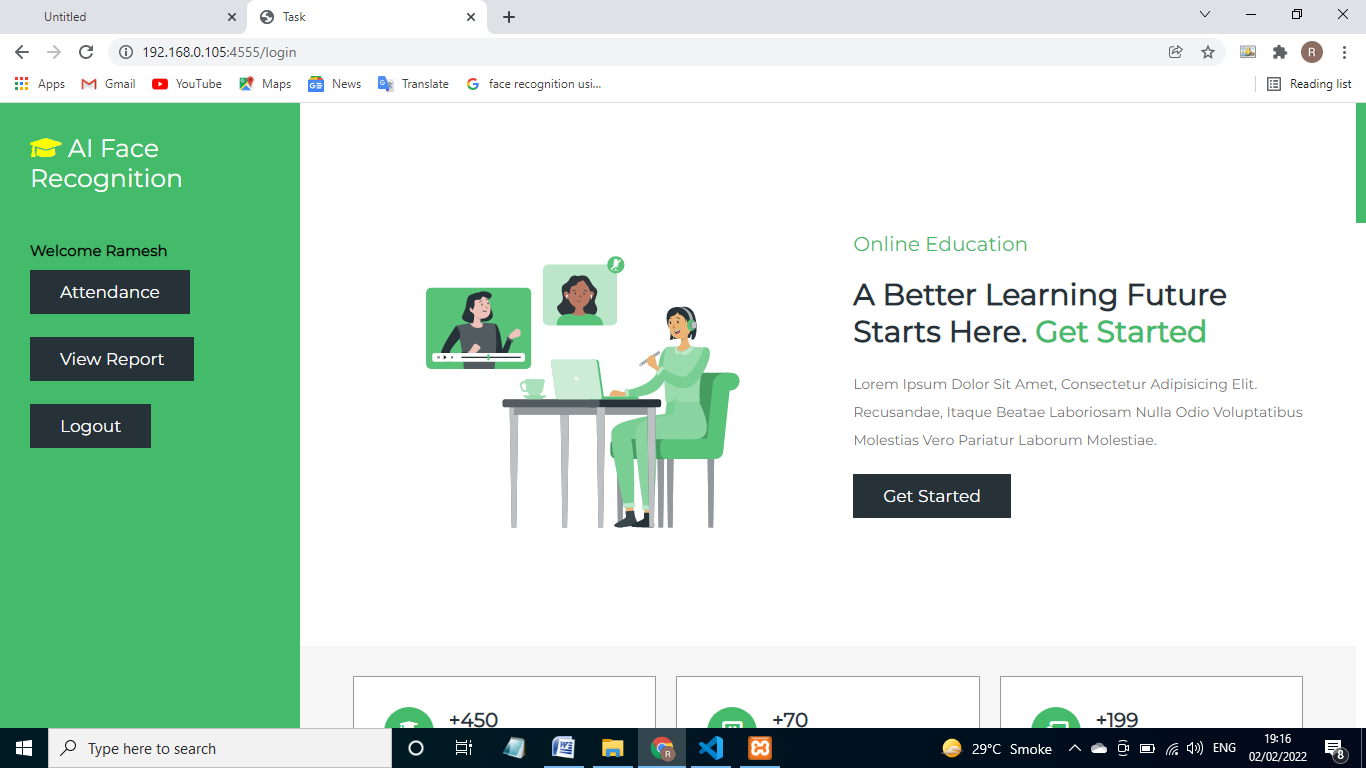
****

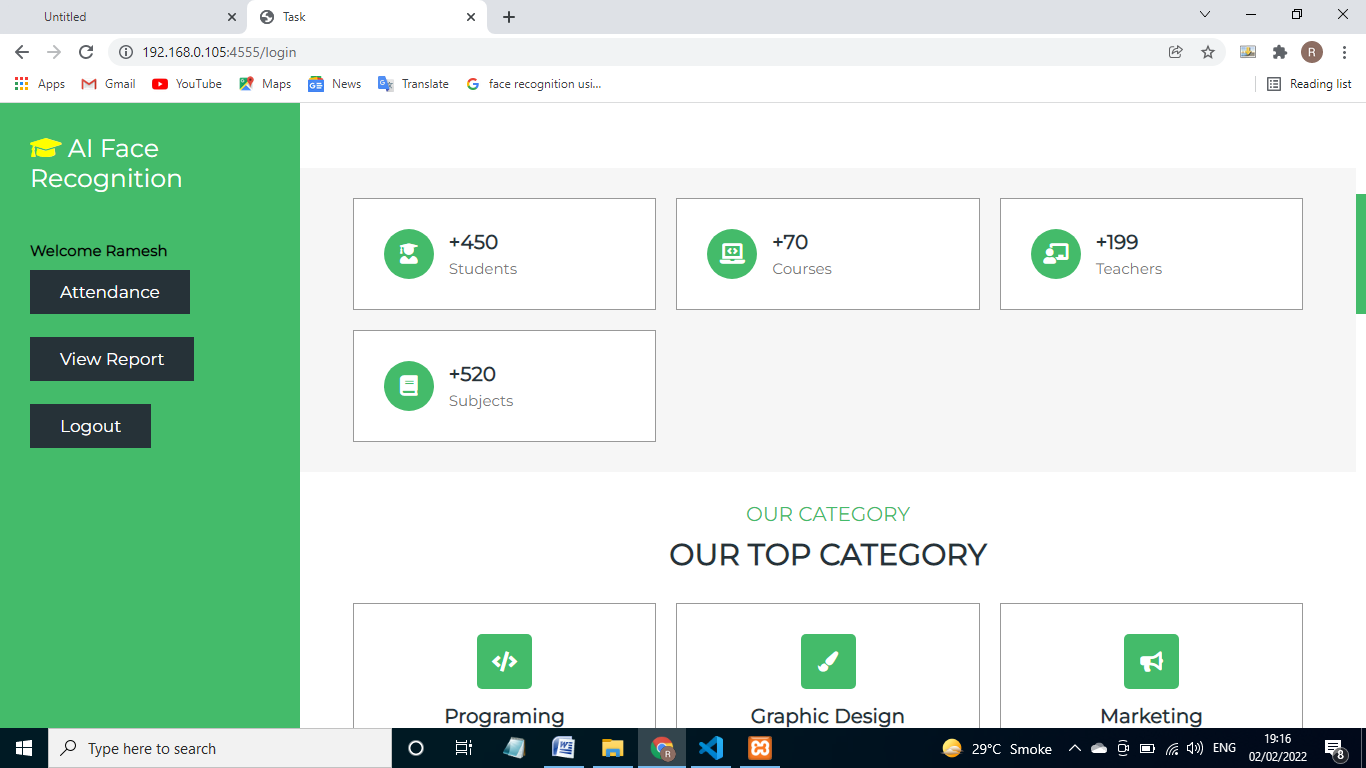
****

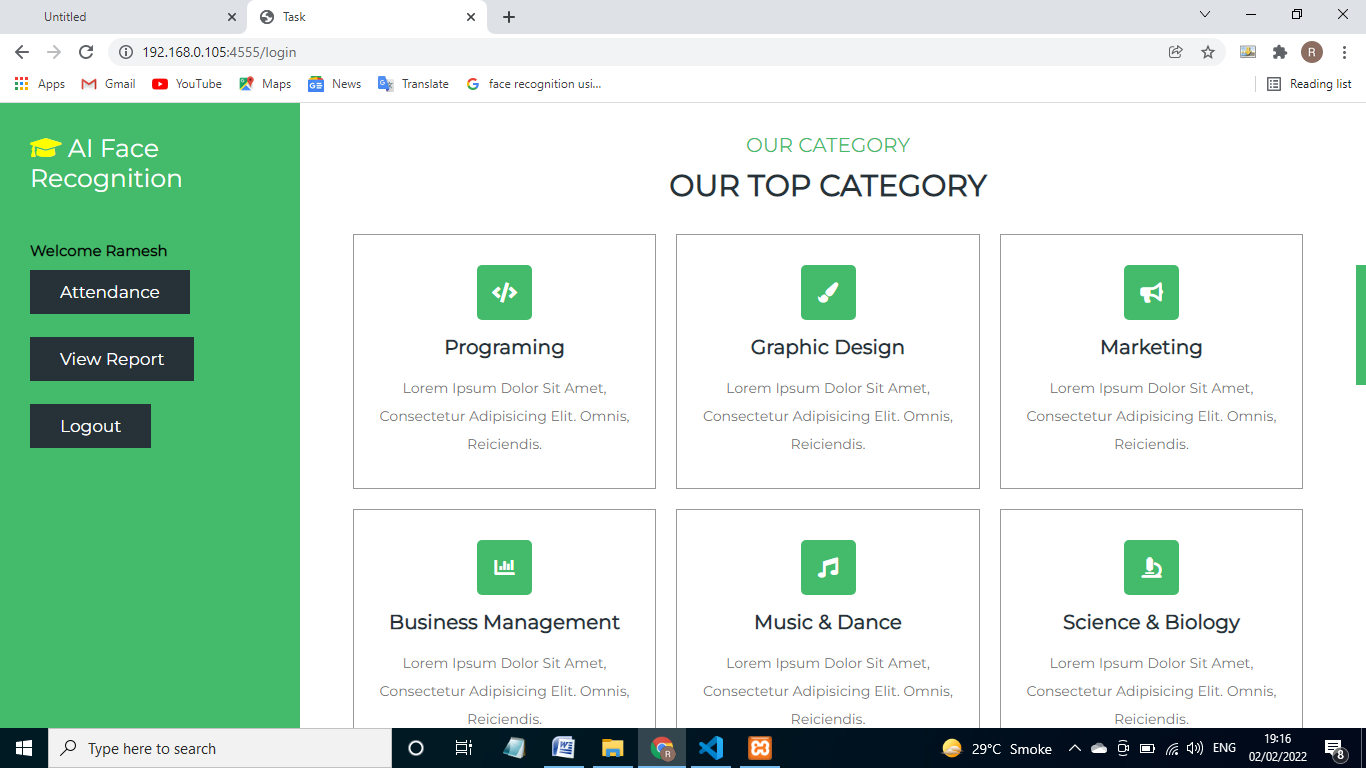
**LoginPage:**

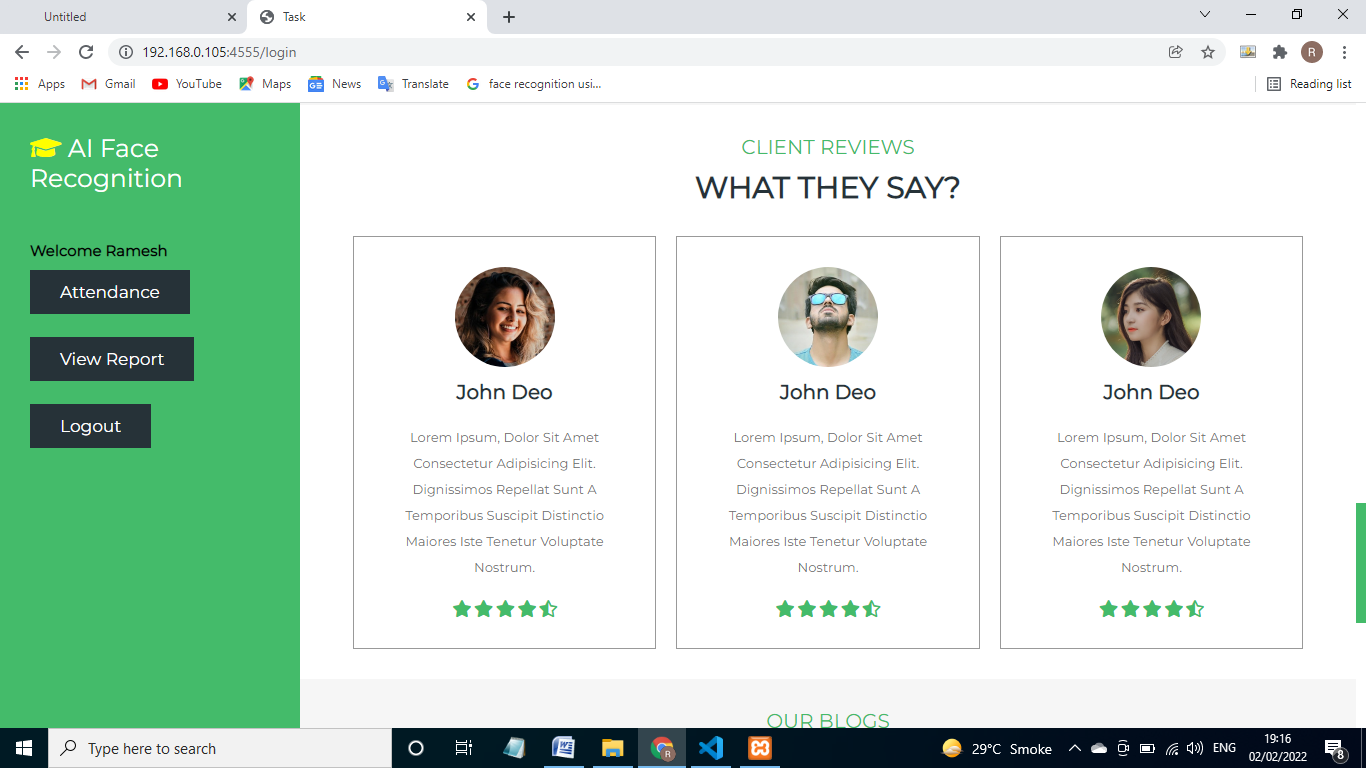
****

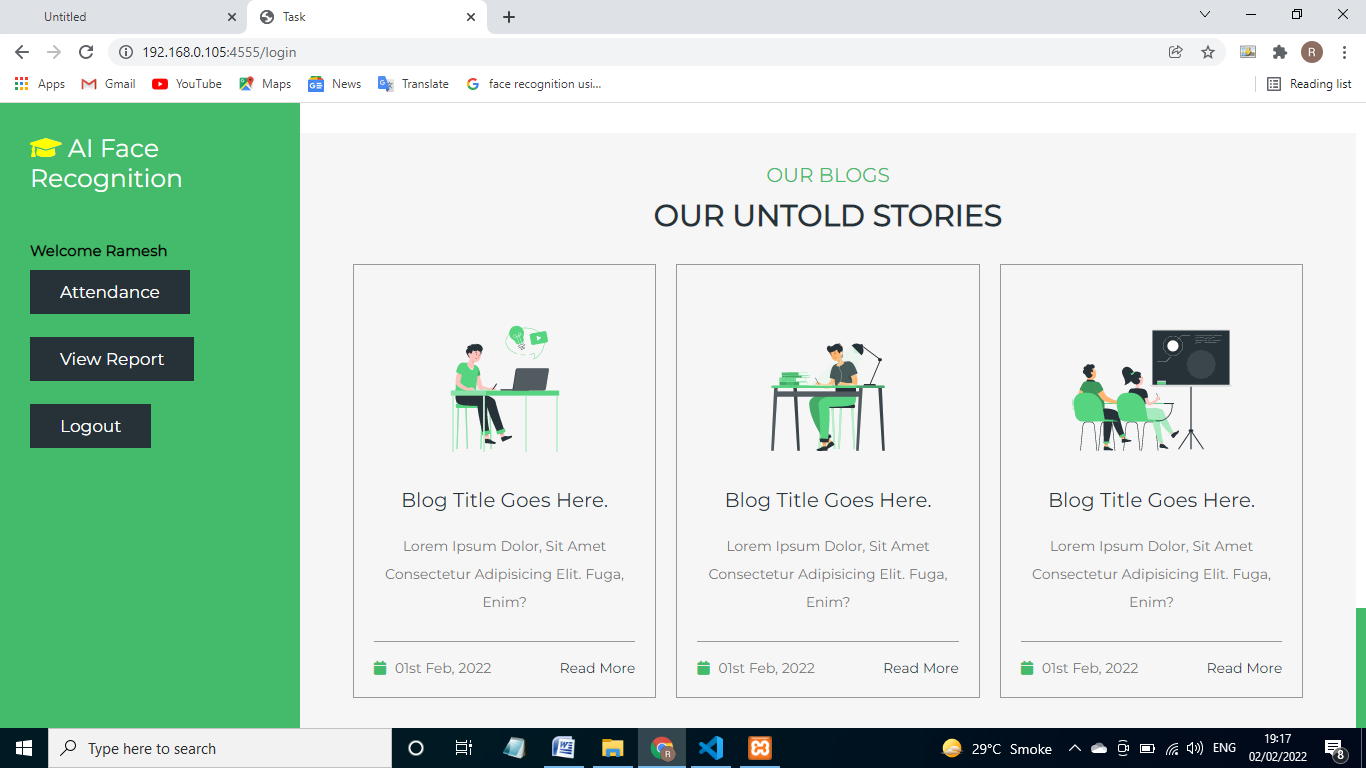
**TaskPage:**

****

****

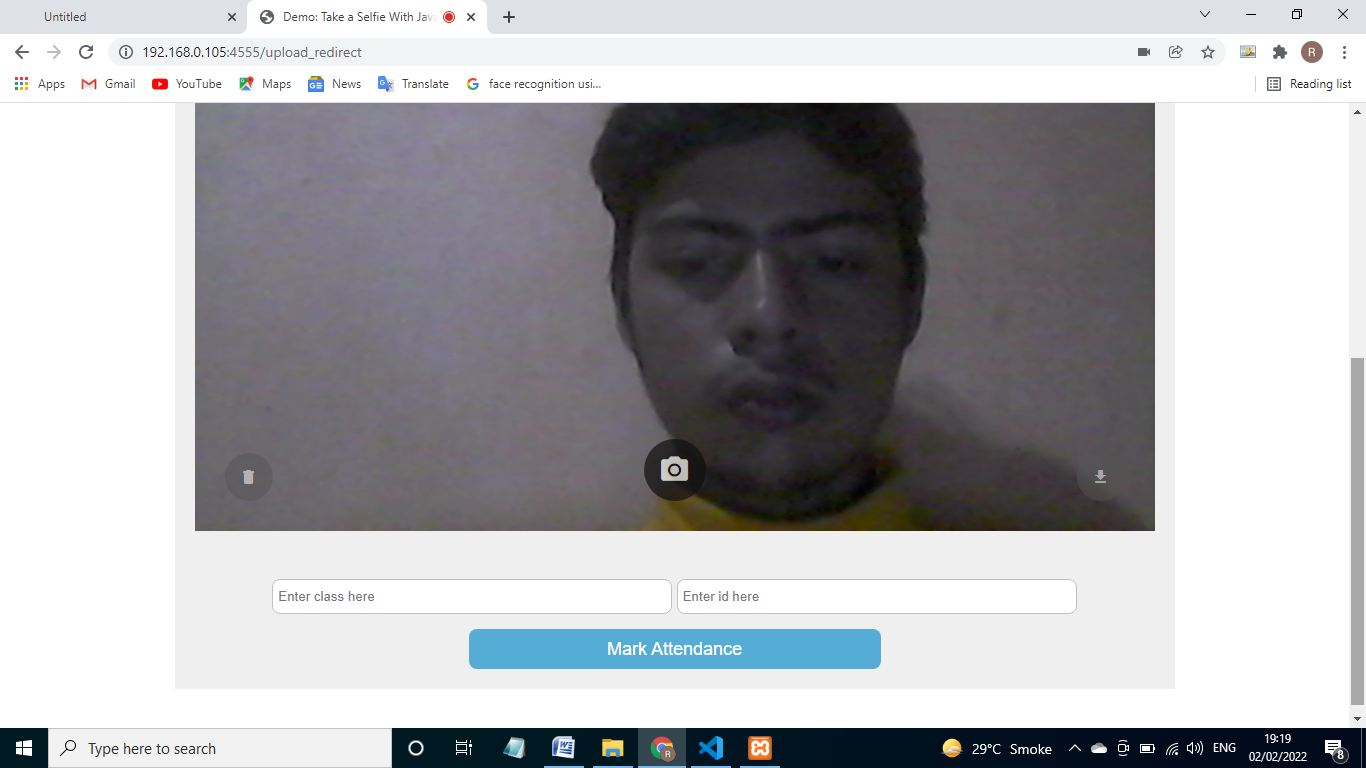
****

****

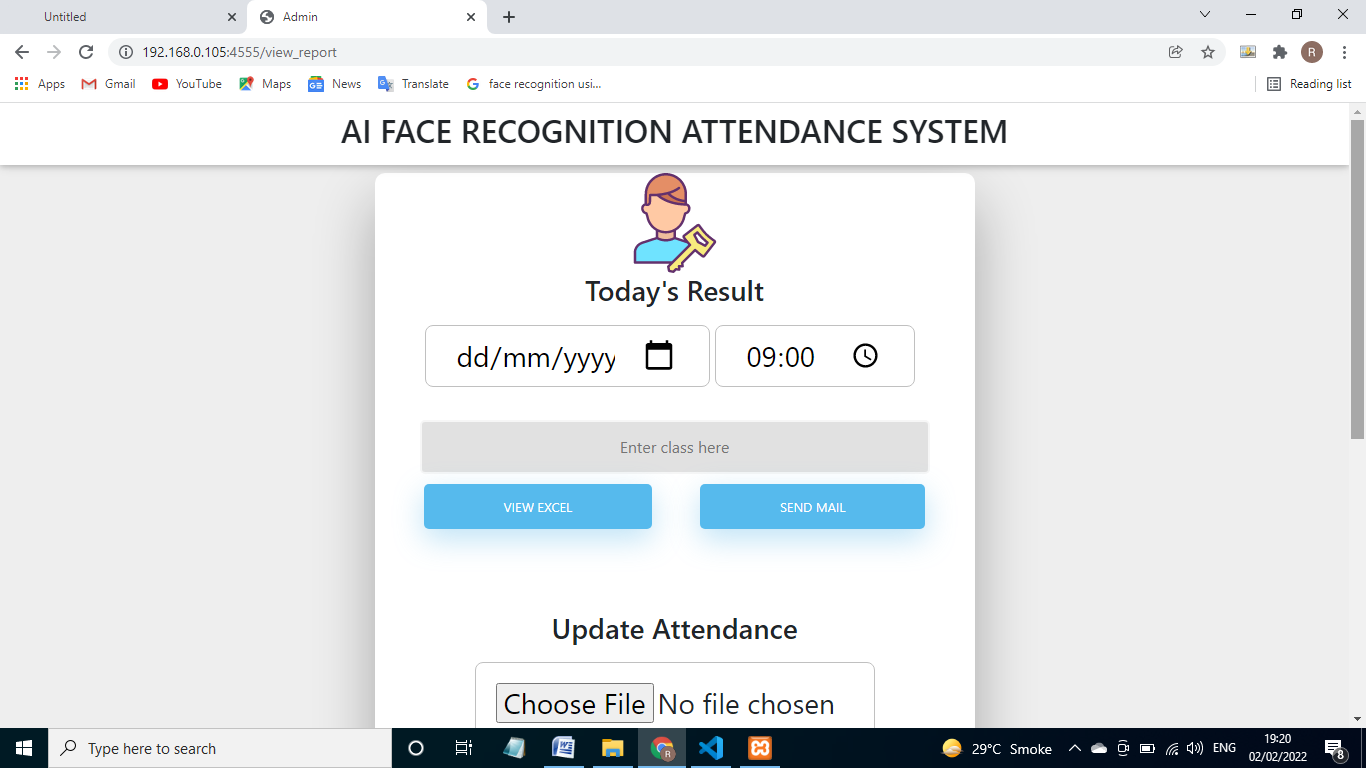
****

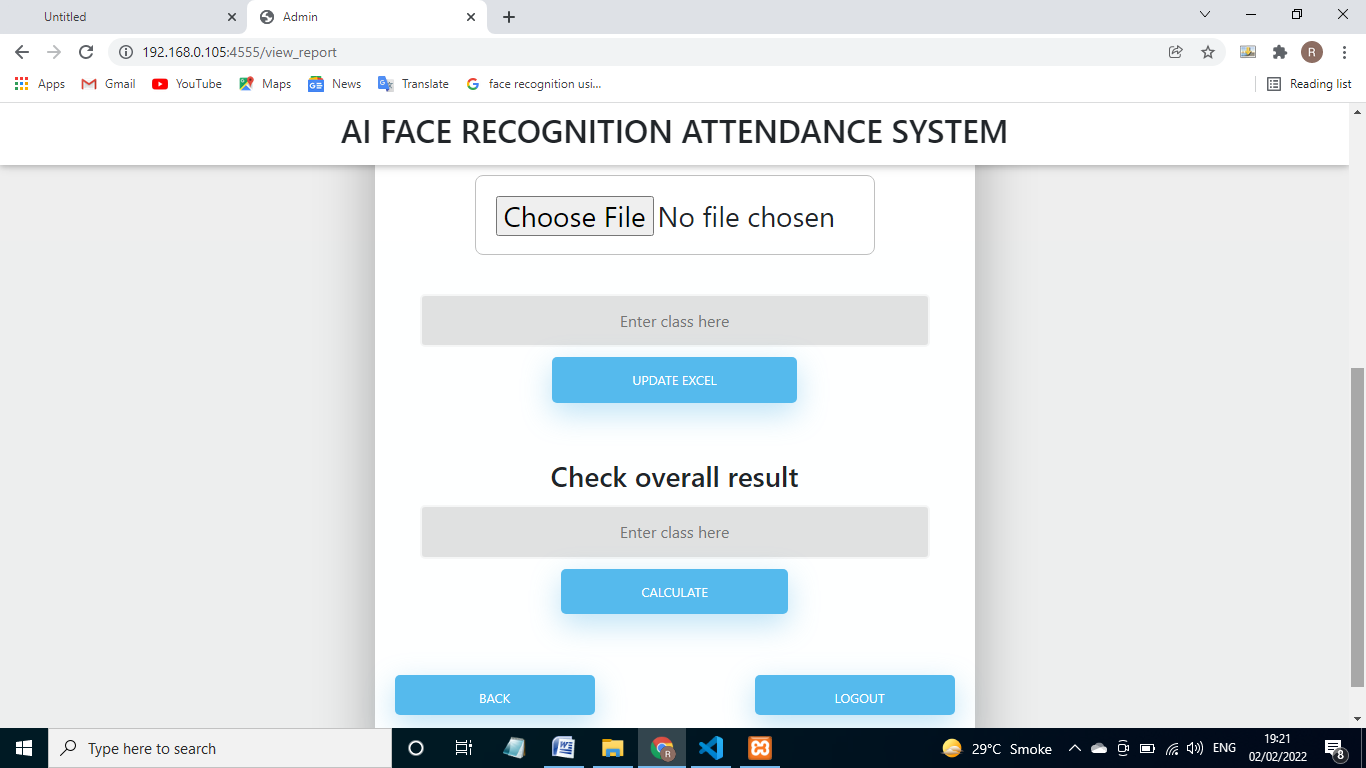
**AttendancePage:**

****

****

**View Report Page:**

****

****

# 6.1 Test Reports