CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1 IMPLEMENTATION

Implementation is the most crucial stage in achieving a successful system and giving the users confidence that the new system is effective and workable. Implementation of this project refers to the installation of the packages in its real environment to the full satisfaction of the users and operations of the system. Testing is done individually at the time of development using the data and verification is done the way specified in the program specification. In short, implementation constitutes all activities that are required to put an already tested and completed package into operation. The success of any information system lies in its successful implementation. System implementation is the stage in the project where the theoretical design is turned into a working system.

5.1.1 ADMIN - app.py

5.1.1.1 PACKAGES

This function will load the necessary packages including math, sklearn, pickle, face_recognition that are needed for admin functionality.

```
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
```

5.1.1.2 AUTHENTICATION SYSTEM

This function will authenticate the admin by verifying the username and password. Once the admin successfully logged in, register_teacher function is called where the faculty details are collected and stored it in the database. Similar to the faculty addition, the student addition can be carried out by calling the function called signup_student. In signup_student, where the basic details and the image of the student is collected to the purpose of model generation.

```
app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath(__file__))
print(APP_ROOT)

conn = MySQLdb.connect(host="localhost",user="root",password="Arasii@5670",db=
"login_info")

@app.route('/')
def index():
    return render_template("index.html",title="Admin Login")
@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():
    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==1):
        return render_template("task.html")
    else:
        return render_template("index.html",title="Admin Login",msg="The username or
password is incorrect")


@app.route('/register_teacher',methods=['POST'])
def register_teacher():
    return render_template("signup.html",title="SignUp")
@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('/student',methods=['POST'])
def file_upload():
    return render_template("upload.html")


@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("signup.html",title="SignUp",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 roll_id=%s",
[roll_id])
    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("/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()
```

5.1.1.3 TRAINING PHASE

This function is responsible for training student data through face recognition and storing it in a database. The whole training process involves preprocessing the image and the model generation.

```
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)
```

5.1.1.4 MODEL GENERATION

This function will result in the model generation through the input image and storing the result in the folder which is named after the register number of the student.

```
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)
```

5.1.2 TEACHERS - APP.PY

5.1.2.1 PACKAGES

This function will load the necessary packages including flask, flask bootstrap, MySQLdb, sklearn, shutil, pickle, face_recognition, xlsxwriter, pandas, flask_mail that are needed for faculty functionality.

```
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
```

5.1.2.2 IMAGE CAPTURING

This function will be responsible for capturing the image of the student to mark the attendance. Here the image is taken and pass it to the match() function where the prediction taken place.

```
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()
```

5.1.2.3 IMAGE DETECTION

This function will classify the student based upon the model which is already trained and stored in database. For prediction, predict() function is called and the classification is done using KNN classifier. Once the prediction is completed, the status of the student is stored in the excel sheet.

```
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
        if f_preds[0][0] == 'unknown':
            return render_template("upload.html",msg= "Student Not Matched")
        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')
        df.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")
```

5.1.2.4 ATTENDANCE VIEW

This function will be responsible for viewing the attendance(i.e) report for the

particular class.

```
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")
```

5.1.2.5 MAIL CONFIGURATION

```
# mail settings

app.config.update(
    DEBUG = True,
    #Email settings
    MAIL_SERVER = 'smtp.gmail.com',
    MAIL_PORT = 465,
    MAIL_USE_SSL = True,
    MAIL_USERNAME = 'demo@gmail.com',
    MAIL_PASSWORD = 'demo12345',
    MAIL_DEFAULT_SENDER = 'demo@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"
```

5.1.2.6 MAIL ALERT

The configuration of the mail server with their credentials to send the alert to the student and parents regarding their attendance status.

```
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>"
```

5.1.2.7 ATTENDANCE UPDATION

```
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")
```

5.1.2.8 CONSOLIDATED REPORT GENERATION

The function to calculate the overall percentage of the particular course or student.

```
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)
```

5.2 SAMPLE OUTPUT

Figure 5.2.1 Test images

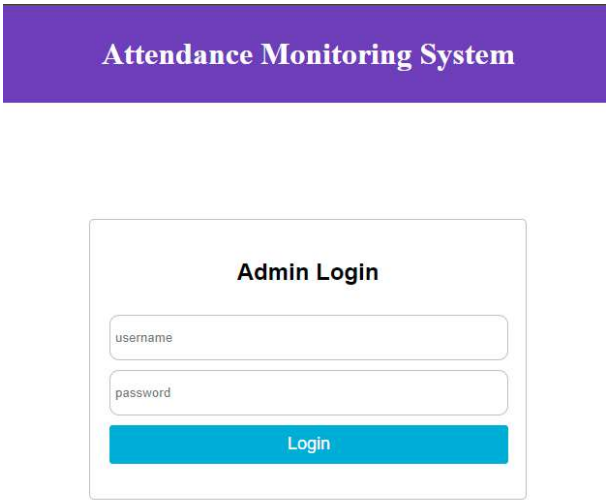


Figure 5.2.2 Admin panel

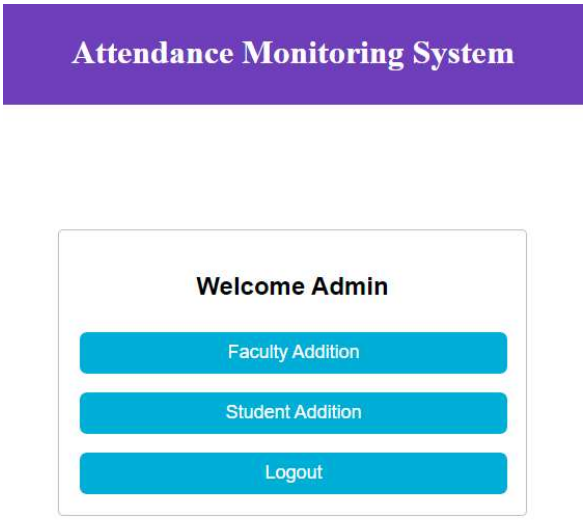


Figure 5.2.3 Admin dashboard

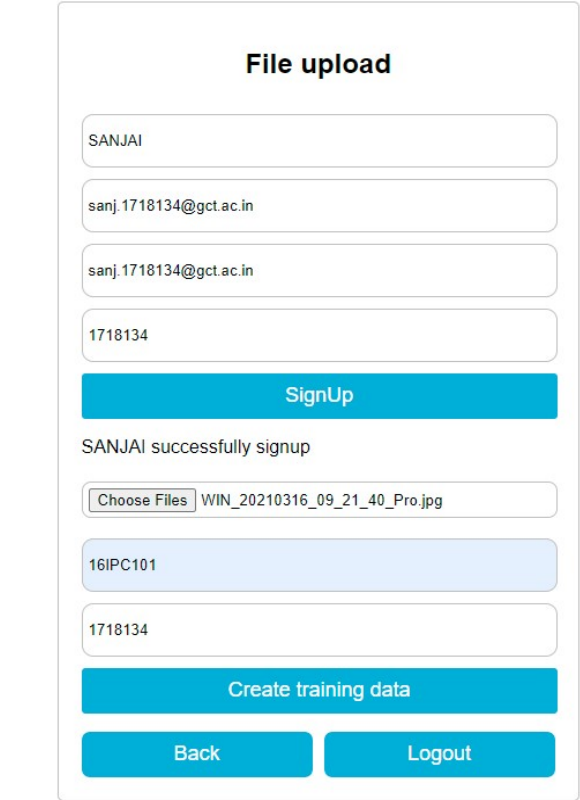
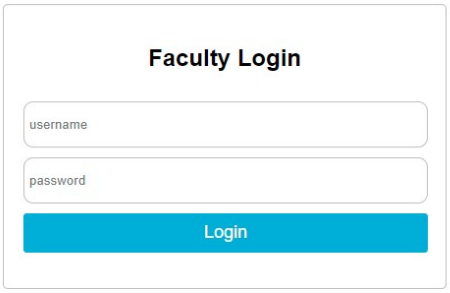


Figure 5.2.4 Student addition



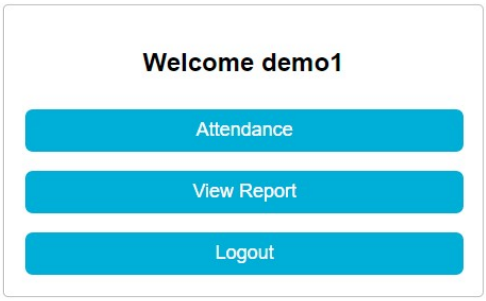
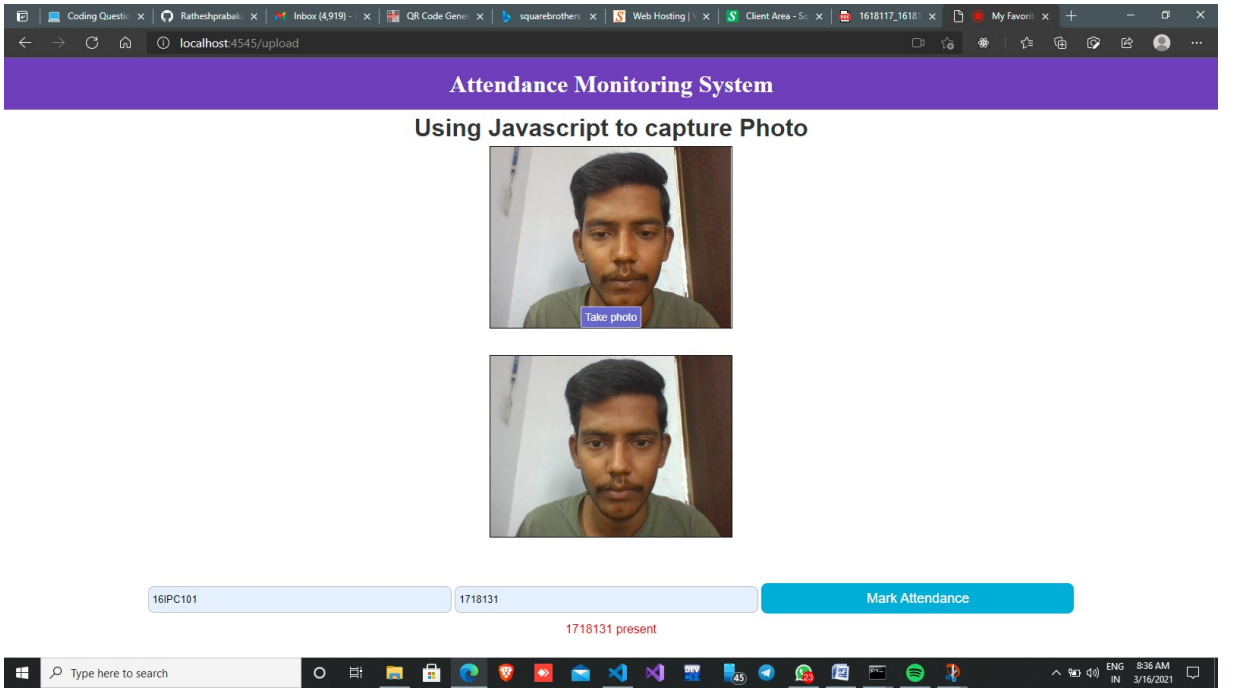


Figure 5.2.5 Faculty login

Figure 5.2.6 Faculty dashboard



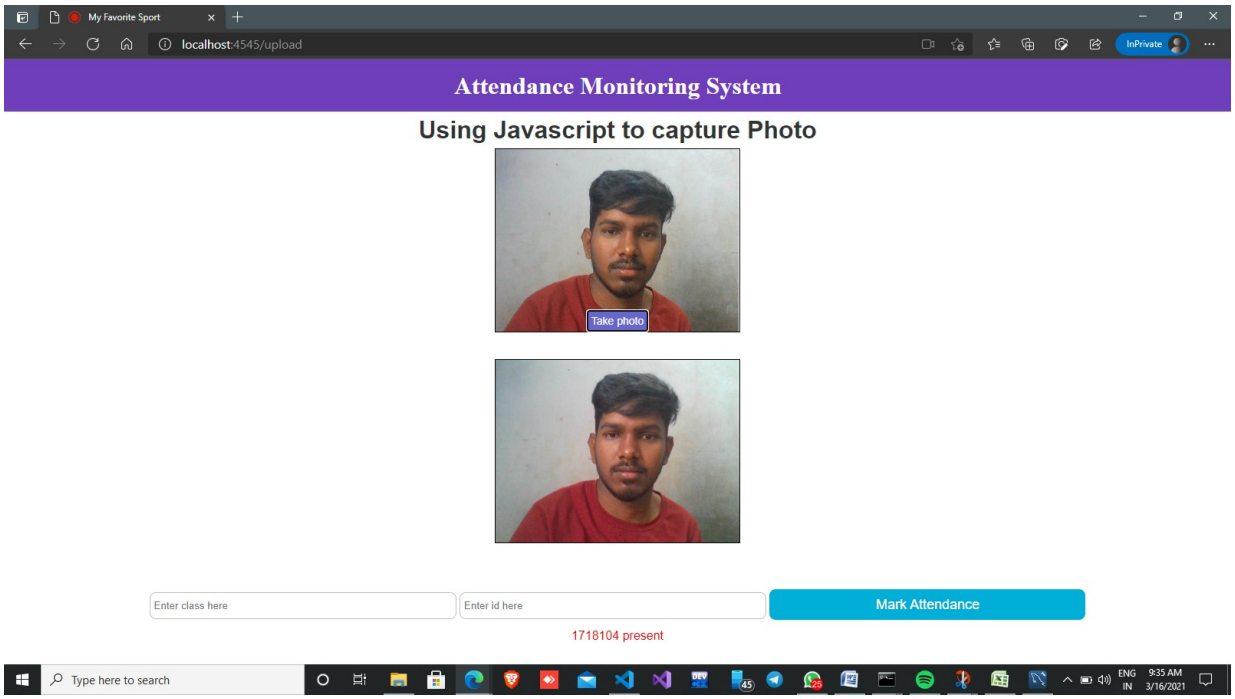


Figure 5.2.7 Take attendance

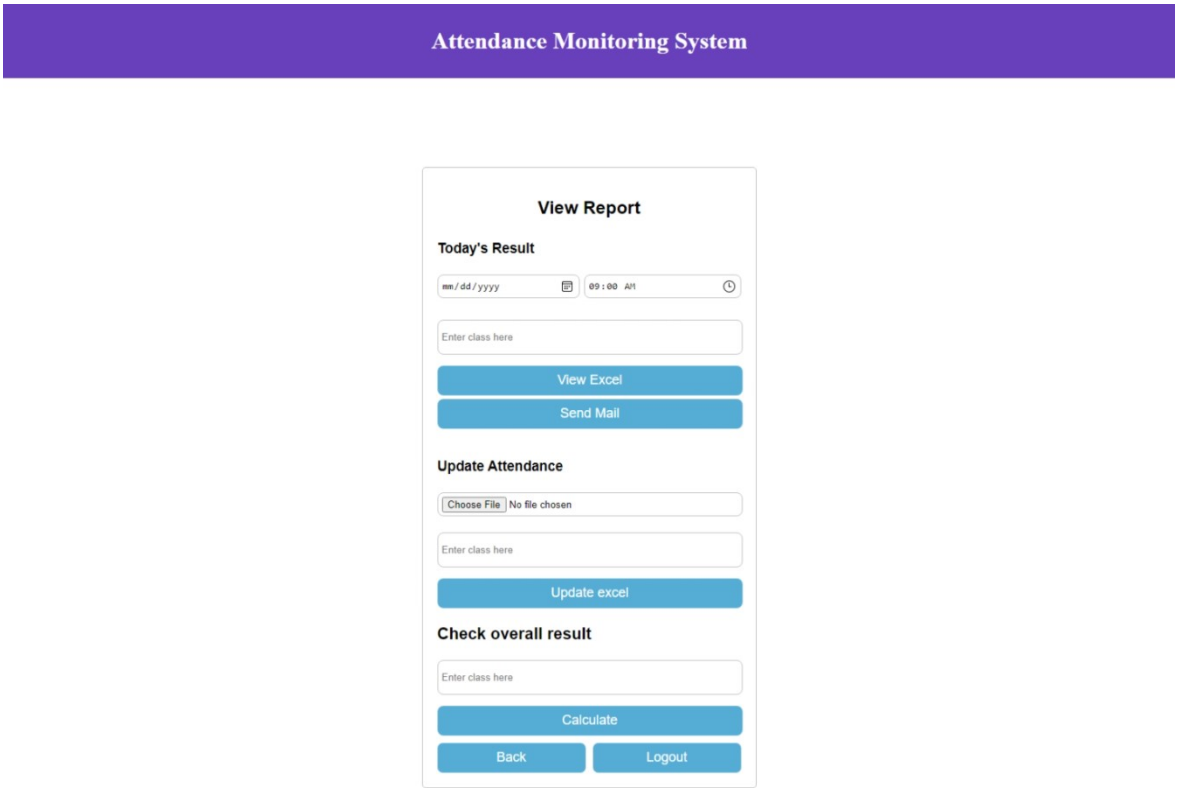


Figure 5.2.8 Report panel

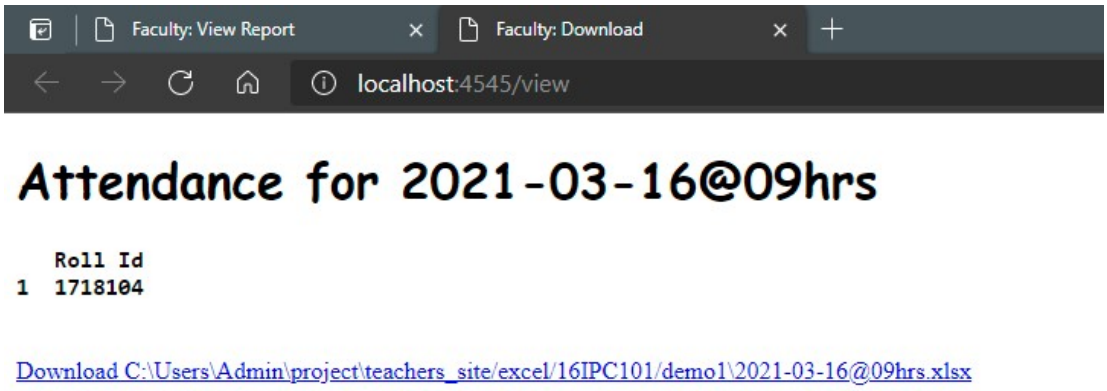


Figure 5.2.9 Attendance view

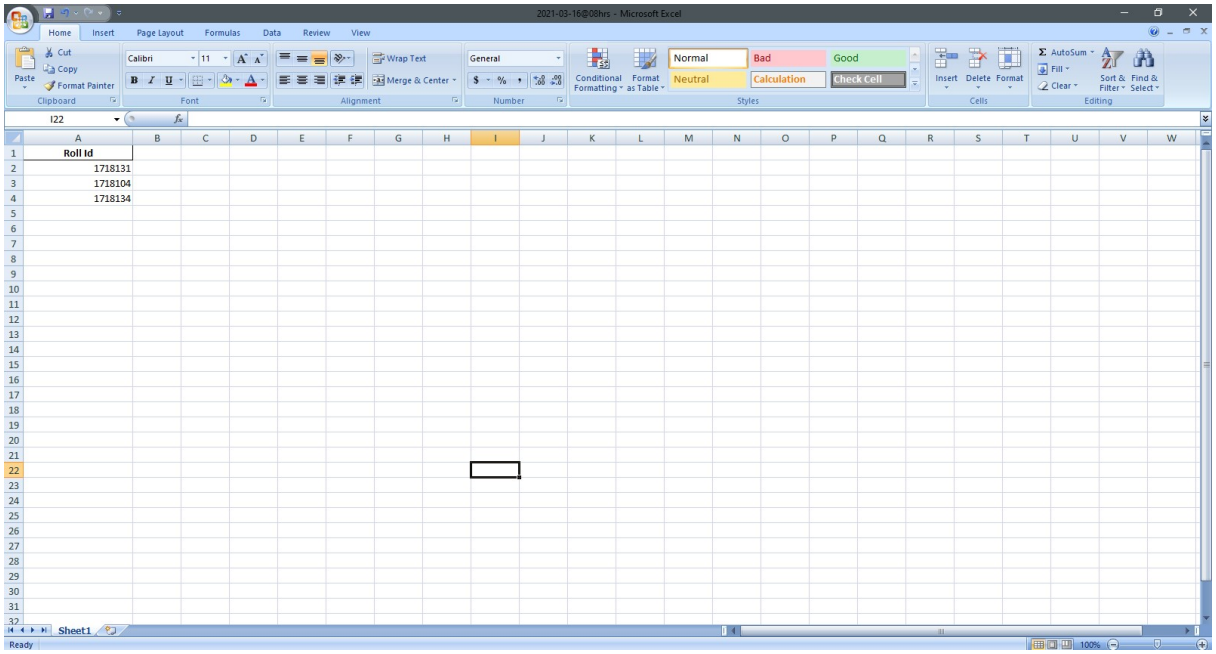


Figure 5.2.10 Report view