ABSTRACT

The Marksheet Management Portal is a comprehensive web-based solution designed to digitize and streamline the academic marksheet management process for educational institutions. Traditional methods of marksheet distribution are often time-consuming, prone to errors, and lack transparency. This portal addresses these challenges by providing a secure, efficient, and user-friendly platform for students, faculty, and administrators.

Developed using Flask (Python), HTML, CSS, and MySQL/SQLite, the portal offers role-based access for students and admins. Students can securely log in to view, download, and track their academic performance, including subject-wise marks, GPA, and result history. The portal supports PDF marksheet downloads, ensuring easy sharing and record-keeping.

Faculty and administrators benefit from an intuitive dashboard that enables efficient management of student records, subjects, exams, and notifications. The system supports bulk result uploads, automated GPA calculation, and real-time publishing of results. Notifications and announcements keep users informed about important updates, exam schedules, and institutional news.

Security is a core focus, with encrypted data storage, secure authentication, and role-based permissions. The portal features a responsive design, ensuring accessibility across desktops, tablets, and smartphones. Additional modules include a robust FAQ/help section, contact/support form, and a password recovery mechanism using OTP verification.

INTRODUCTION

The Marksheet Management Portal is a full-stack web application developed to modernize and simplify the academic marksheet management process for colleges and educational institutions. Built using Python's Flask framework for the backend and HTML, CSS, and JavaScript for the frontend, the portal provides a secure, efficient, and user-friendly platform for students, faculty, and administrators. The system connects to a MySQL database to store and manage all academic records, including student profiles, subjects, exams, marks, and notifications.

Students can securely log in to view their subject-wise marks, GPA, and download their marksheets in PDF format. The portal also allows students to request rechecking of marks, view result history, and receive important notifications and announcements. Admins have access to a dedicated dashboard where they can manage student records, upload marks in bulk, handle rechecking requests, and send notifications to students and faculty.

Security is a core focus, with OTP-based login and password reset, encrypted data storage, and role-based access control. The portal features a responsive design, ensuring accessibility on desktops, tablets, and smartphones. Additional modules include a feedback system, FAQ/help section, and a contact form for support. The Marksheet Management Portal enhances transparency, reduces administrative workload, and empowers users with instant access to academic information, making it an ideal solution for digital result management in educational institutions.

LITERATURE SURVEY

The management of academic marksheets and student records has traditionally relied on manual processes, involving paper-based documentation and physical distribution. This approach is often time-consuming, error-prone, and lacks transparency, leading to delays and difficulties in accessing academic information. With the advancement of technology, educational institutions have increasingly adopted digital solutions to address these challenges.

Several existing systems, such as university ERP platforms and online result portals, provide functionalities for result publication and student record management. However, many of these systems are either proprietary, costly, or lack user-friendly interfaces tailored for both students and administrators. Open-source solutions like Fedena and EduSec offer comprehensive modules but may require significant customization and technical expertise for deployment.

Recent research and case studies highlight the benefits of web-based marksheet management systems, including improved accessibility, enhanced data security, and streamlined administrative workflows. Features such as online result viewing, downloadable marksheets, automated GPA calculation, and notification modules have become standard expectations in modern academic portals.

The Marksheet Management Portal builds upon these advancements by integrating secure login, OTP-based password recovery, role-based access, and a responsive user interface. Unlike many generic solutions, this portal is specifically designed for ease of use, transparency, and scalability in college environments. By incorporating feedback, help, and contact modules, the system ensures comprehensive support for all users.

PROBLEM STATEMENT

Despite advancements in digital infrastructure, many educational institutions still rely on manual or semi-digital processes for managing and distributing academic marksheets. This traditional approach introduces several challenges that hinder efficiency, transparency, and user satisfaction for both students and administrators. The key problems addressed by this project are:

- **Time-Consuming Processes:** Manual entry, verification, and distribution of marksheets lead to significant delays in result publication and access.
- **Prone to Errors:** Human errors in data entry and calculation can result in incorrect marksheets, causing confusion and dissatisfaction among students.
- **Lack of Transparency:** Students often have limited visibility into their academic records and rechecking status, leading to repeated inquiries and administrative overhead.
- **Data Security Risks:** Physical and unencrypted digital records are vulnerable to unauthorized access, loss, or tampering.
- **Limited Accessibility:** Students and faculty may face difficulties accessing marksheets and notifications remotely, especially during holidays or emergencies.
- **Inefficient Communication:** Important announcements and result notifications may not reach all stakeholders promptly, resulting in missed information.
- **Cumbersome Rechecking Process:** Requesting and tracking rechecking or corrections in marksheets is often slow and lacks a systematic workflow.

SOFTWARE DEVELOPMENT

- **Requirement Analysis**

  The initial phase involved gathering requirements from students, faculty, and administrators to understand their needs regarding marksheet access, result publication, notifications, and security. This helped in defining clear objectives and essential features for the portal.

- **System Design**

  Based on the requirements, the system architecture was designed using Flask for the backend, MySQL/SQLite for the database, and HTML/CSS/JavaScript for the frontend. The design emphasized modularity, security, and ease of use, with separate modules for authentication, result management, notifications, and support.

- **Implementation**

  The portal was developed using Python's Flask framework, integrating Jinja2 templating for dynamic content rendering. Database models were created for students, admins, subjects, exams, marks, and notifications. Responsive UI components were built to ensure accessibility across devices.

- **Testing**

  Comprehensive testing was conducted by simulating real-world scenarios such as student login, marksheet download, admin result uploads, password recovery, and notification delivery. Both valid and invalid user actions were tested to ensure robust error handling and data integrity.

- **Deployment**

  The Marksheet Management Portal was initially deployed and tested in a local development environment to ensure all functionalities worked as intended. For production and public access, the application was deployed on Render, a cloud-based platform that supports Python web applications. Render provides automated builds, easy scaling, and secure hosting, making it suitable for educational projects.

- **Maintenance & Future Enhancements**

  The portal is designed for easy maintenance, allowing for bug fixes, feature enhancements, and adaptation to user feedback. Planned future enhancements include integration with SMS/email notifications, advanced analytics, multi-language support, and an expanded admin dashboard

SOURCE CODE

```
from flask import Flask, render_template, request, redirect, session, url_for, flash, jsonify, make_response, send_file # Flask imports
import os  # For file handling
import mysql.connector  # MySQL Connector
from flask_mail import Mail, Message
import random  # For generating OTP
import  re #use for regex validation
import datetime  # For date and time handling
from datetime import timedelta # For session expiration
import pdfkit  #used for pdf generation


app = Flask(__name__)
app.secret_key = 'S@rb3sw@r_OTP_Se$$ion_Key_123!'  # Replace with a strong secret


app.permanent_session_lifetime = timedelta(days=2)

# Flask-Mail Configuration
app.config.update(
    MAIL_SERVER='smtp.gmail.com',
    MAIL_PORT=587,
    MAIL_USE_TLS=True,
    MAIL_USERNAME='verify.marksheet.manage.portal@gmail.com',       # Replace with your email
    MAIL_PASSWORD='redcvumsuuigdgra'          # Replace with your app password
)
mail = Mail(app)

# MySQL DB Connection
def get_db_connection():
    return mysql.connector.connect(
        host='mysql-d8d8c8a-sarbeswarpanda143-36cb.f.aivencloud.com',
        user='avnadmin',
        port=25309,
        password='AVNS__Nh9xl2rlVgCUarubfp',              # Your MySQL root password
        database='marksheet_portal'            # Your database name
    )

# ---------------- LOGIN ----------------
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        print("Login POST received")

        user_type = request.form.get('role')  # 'student' or 'admin'
        username = request.form.get('username')
        password = request.form.get('password')

        print("Username:", username)
        print("Password:", password)

        # Student validation
        if user_type == 'student':
            if not (username.isdigit() and len(username) == 10):
                flash("Registration number must be a 10-digit number.", 'danger')
                return redirect(url_for('login'))

        # Password pattern validation
        if not re.match(r'^(?=.*[a-zA-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,}$', password):
            flash("Password must be alphanumeric with at least one special character.", 'danger')
            return redirect(url_for('login'))

        # Connect to DB and validate user
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        user = None
```

```
if user_type == 'student':
  query = "SELECT * FROM students WHERE registration_number = %s AND password = %s"
  print("Executing query:", query, "with params:", (username, password))
  cursor.execute(query, (username, password))
  user = cursor.fetchone()
elif user_type == 'admin':
  query = "SELECT * FROM admin WHERE admin_id = %s AND password = %s"
  print("Executing query:", query, "with params:", (username, password))
  cursor.execute(query, (username, password))
  user = cursor.fetchone()
else:
    flash("Invalid user type.", "danger")
    return redirect(url_for('login'))

print("User from DB:", user)

cursor.close()
conn.close()

if user:
    session['username'] = username
    session['user_type'] = user_type
    session['email'] = user['email']

          # Add role-specific identifiers
    if user_type == 'admin':
        session['admin_id'] = user['admin_id']  # Store admin_id for admin users
    elif user_type == 'student':
        session['student_id'] = user['id']  # Store student_id for student users


    # --- Remember Me logic START ---
    if 'remember_me' in request.form:
        session.permanent = True  # Session will use app.permanent_session_lifetime
    else:
        session.permanent = False

    otp = str(random.randint(100000, 999999))
    session['otp'] = otp

    print("Session values:", dict(session))

    try:
        msg = Message('Your OTP for Login', sender=app.config['MAIL_USERNAME'], recipients=[user['email']])
        msg.html = f"""
        <!DOCTYPE html>
        <html>
        <head>
          <style>
            body {{
              font-family: 'Arial', sans-serif;
              background-color: #f4f4f4;
              color: #333;
              padding: 20px;
              margin: 0;
              font-size: 16px;
              line-height: 1.5;
              text-align: center;

            }}
            .container {{
              max-width: 600px;
              margin: 0 auto;
              background: #ffffff;
              padding: 20px;
              border-radius: 8px;
              box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

            }}
            h1 {{
              color: #007bff;
              font-size: 2rem;
```

```
                margin-bottom: 20px;
            }}
            p {{
                line-height: 1.6;
                font-size: 1rem;
                margin: 10px 0;
            }}
            .otp {{
                font-size: 1.5rem;
                font-weight: bold;
                color: #28a745;
                margin: 20px 0;
            }}
            .footer {{
                margin-top: 20px;
                font-size: 0.9rem;
                color: #666;
                text-align: center;
            }}
        </style>
    </head>
    <body>
        <div class="container">
            <h1>Marksheet Management Portal</h1>
            <p>Dear {user['name']},</p>
            <p>We have received a login request for your account. To proceed, please use the following One-Time Password (OTP):</p>
            <p class="otp">{otp}</p>
            <p>This OTP is valid for the next 10 minutes. If you did not request this, please ignore this email or contact support immediately.</p>
            <p>Thank you,<br>The Marksheet Management Portal Team</p>
            <div class="footer">
                <p>© 2025 Marksheet Management Portal. All rights reserved.</p>
            </div>
        </div>
    </body>
    </html>
    """
    # Attach a file (e.g., a PDF or image)
    file_path = "static/images/hero-bg.jpg"  # Path to the file you want to attach
    if os.path.exists(file_path):
        with app.open_resource(file_path) as attachment:
            msg.attach("hero-bg.jpg", "application/pdf", attachment.read())
    else:
        print("Attachment file not found:", file_path)

    mail.send(msg) # Send the email with the OTP

    flash('OTP sent to your email.', 'info')  # Use 'info' for OTP sent message
    print("✅ Email sent successfully.")
    return redirect(url_for('verify_otp'))  # ✅ MISSING RETURN ADDED HERE!
  except Exception as e:
    print("❌ Failed to send email:", e)
    flash('Failed to send OTP. Please try again later.', 'danger')
    return redirect(url_for('login'))

else:
  flash('Invalid credentials. Try again.', 'danger')
  return redirect(url_for('login'))

return render_template('login.html')


@app.route('/verify_otp', methods=['GET', 'POST'])
def verify_otp():
  if request.method == 'POST':
    entered_otp = request.form.get('otp_input')
    session_otp = session.get('otp')

    if entered_otp == session_otp:
      flash('OTP verified successfully!', 'success')
```

```
        # Redirect based on role
        if session.get('user_type') == 'admin':
            return redirect(url_for('admin_dashboard'))
        else:
            return redirect(url_for('student_dashboard'))
    else:
        flash('Incorrect OTP. Please try again.', 'danger')
        return redirect(url_for('verify_otp'))

  return render_template('verify_otp.html')

@app.route('/resend_otp')
def resend_otp():
  # Only allow if user is in session
  if 'email' not in session or 'username' not in session:
    flash('Session expired. Please login again.', 'danger')
    return redirect(url_for('login'))

  otp = str(random.randint(100000, 999999))
  session['otp'] = otp

  try:
    msg = Message('Your OTP for Login', sender=app.config['MAIL_USERNAME'], recipients=[session['email']])
    msg.body = f"Your new OTP is: {otp}"
    mail.send(msg)
    flash('A new OTP has been sent to your email.', 'info')
  except Exception as e:
    print("❌ Failed to send email:", e)
    flash('Failed to send OTP. Please try again later.', 'danger')

  return redirect(url_for('verify_otp'))

@app.route('/verify_reset_otp', methods=['GET', 'POST'])
def verify_reset_otp():
  if request.method == 'POST':
    entered_otp = request.form.get('otp')
    if entered_otp == session.get('reset_otp'):
      flash("OTP verified! Set your new password.", "success")
      return redirect(url_for('reset_password'))
    else:
      flash("Incorrect OTP. Try again.", "danger")
      return redirect(url_for('verify_reset_otp'))
  else:
    # This is your resend OTP logic
    otp = str(random.randint(100000, 999999))
    session['reset_otp'] = otp
    user_email = session.get('reset_email')
    if user_email:
      try:
        msg = Message('Your New OTP', sender=app.config['MAIL_USERNAME'], recipients=[user_email])
        msg.body = f'Your new OTP is: {otp}'
        mail.send(msg)
        flash('A new OTP has been sent to your email.', 'success')
      except Exception as e:
        flash('Failed to send OTP. Please try again.', 'danger')
    else:
      flash('Session expired. Please login again.', 'danger')
      return redirect(url_for('login'))
    return render_template('verify_reset_otp.html')



# ---------- DASHBOARDS ----------

@app.route('/student_dashboard')
def student_dashboard():
  # Ensure the user is logged in and is a student
  if 'student_id' not in session or session.get('user_type') != 'student':
    flash('Unauthorized access. Please log in as a student.', 'danger')
    return redirect(url_for('login'))

  conn = get_db_connection()
```

```
    cursor = conn.cursor(dictionary=True)

    # Fetch student details
    cursor.execute("SELECT * FROM students WHERE id = %s", (session['student_id'],))
    student = cursor.fetchone()

    # Fetch exam records for the student (with exam and subject info)
    cursor.execute("""
        SELECT marks.id, exams.id AS exam_id, exams.name AS exam_name,
               subjects.id AS subject_id, subjects.name AS subject_name,
               marks.marks_obtained
        FROM marks
        JOIN exams ON marks.exam_id = exams.id
        JOIN subjects ON marks.subject_id = subjects.id
        WHERE marks.student_id = %s
        ORDER BY exams.id, subjects.name
    """, (student['id'],))
    records = cursor.fetchall()

    # Build exams_dict for accordion
    exams_dict = {}
    for row in records:
        exam_id = row['exam_id']
        if exam_id not in exams_dict:
            exams_dict[exam_id] = {
                'exam_name': row['exam_name'],
                'subjects': []
            }
        exams_dict[exam_id]['subjects'].append(row)

    # Fetch rechecking requests for this student
    cursor.execute("""
        SELECT r.status, r.remark, e.name as exam_name, sub.name as subject_name, r.reason, r.solved_at
        FROM rechecking_requests r
        JOIN exams e ON r.exam_id = e.id
        JOIN subjects sub ON r.subject_id = sub.id
        WHERE r.student_id=%s
        ORDER BY r.created_at DESC
    """, (student['id'],))
    student_rechecking_requests = cursor.fetchall()


    # Calculate overall percentage
    cursor.execute("SELECT AVG(marks_obtained) AS percentage FROM marks WHERE student_id = %s", (student['id'],))
    percentage = cursor.fetchone()['percentage'] or 0

    conn.close()

    return render_template(
        'student_dashboard.html',
        student=student,
        exams_dict=exams_dict,
        percentage=round(percentage, 2),
        student_rechecking_requests=student_rechecking_requests
    )

@app.route('/apply-rechecking', methods=['POST'])
def apply_rechecking():
    if 'student_id' not in session:
        flash('Please log in.', 'danger')
        return redirect(url_for('login'))
    student_id = session['student_id']
    exam_id = request.form['exam_id']
    subject_id = request.form['subject_id']
    reason = request.form['reason']
    conn = get_db_connection()
    cursor = conn.cursor()
    # Check if marks are already 100
    cursor.execute("SELECT marks_obtained FROM marks WHERE student_id=%s AND exam_id=%s AND subject_id=%s", (student_id,
exam_id, subject_id))
    result = cursor.fetchone()
    if result and result[0] == 100:
```

```
        flash('You already have 100 marks in this subject. Rechecking not allowed.', 'warning')
        conn.close()
        return redirect(url_for('student_dashboard'))
    # Prevent duplicate requests
    cursor.execute("SELECT id FROM rechecking_requests WHERE student_id=%s AND exam_id=%s AND subject_id=%s AND
status='Pending'", (student_id, exam_id, subject_id))
    if cursor.fetchone():
        flash('Already applied for rechecking for this subject.', 'warning')
    else:
        cursor.execute(
            "INSERT INTO rechecking_requests (student_id, exam_id, subject_id, reason) VALUES (%s, %s, %s, %s)",
            (student_id, exam_id, subject_id, reason)
        )
        conn.commit()
        flash('Rechecking request submitted.', 'success')
    conn.close()
    return redirect(url_for('student_dashboard'))

@app.route('/admin/rechecking-requests')
def admin_rechecking_requests():
    if 'admin_id' not in session:
        return redirect(url_for('login'))
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    # Pending
    cursor.execute("""
        SELECT r.id, r.student_id, s.name as student_name, e.name as exam_name, sub.name as subject_name, r.reason
        FROM rechecking_requests r
        JOIN students s ON r.student_id = s.id
        JOIN exams e ON r.exam_id = e.id
        JOIN subjects sub ON r.subject_id = sub.id
        WHERE r.status='Pending'
        ORDER BY r.created_at DESC
    """)
    pending_requests = cursor.fetchall()
    # Solved
    cursor.execute("""
        SELECT r.id, r.student_id, s.name as student_name, e.name as exam_name, sub.name as subject_name, r.remark, r.reason,
r.solved_at
        FROM rechecking_requests r
        JOIN students s ON r.student_id = s.id
        JOIN exams e ON r.exam_id = e.id
        JOIN subjects sub ON r.subject_id = sub.id
        WHERE r.status='Solved'
        ORDER BY r.solved_at DESC
    """)
    solved_requests = cursor.fetchall()
    conn.close()
    return render_template('admin_rechecking_requests.html',
                           pending_requests=pending_requests,
                           solved_requests=solved_requests)

@app.route('/admin/solve-rechecking', methods=['POST'])
def solve_rechecking():
    if 'admin_id' not in session:
        return redirect(url_for('login'))
    request_id = request.form['request_id']
    remark = request.form['remark']
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("UPDATE rechecking_requests SET status='Solved', remark=%s, solved_at=NOW() WHERE id=%s", (remark,
request_id))
    conn.commit()
    conn.close()
    flash('Marked as solved.', 'success')
    return redirect(url_for('admin_rechecking_requests'))


@app.route('/edit-admin-profile', methods=['GET', 'POST'])
def edit_admin_profile():
    # Ensure the admin is logged in
    if 'admin_id' not in session:
```

```
        flash('Please log in to access this feature.', 'danger')
        return redirect(url_for('login'))

    admin_id = session['admin_id']  # Get the logged-in admin's ID from the session

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    if request.method == 'POST':
        # Get updated details from the form
        name = request.form['name']
        email = request.form['email']
        mobile = request.form['mobile']
        password = request.form['password']

        # Update the admin details in the database
        cursor.execute("""
            UPDATE admin
            SET name = %s, email = %s, mobile = %s, password = %s
            WHERE admin_id = %s
        """, (name, email, mobile, password, admin_id))
        conn.commit()

        flash('Profile updated successfully!', 'success')
        return redirect(url_for('admin_dashboard'))

    # Fetch the admin's current details for the form
    cursor.execute("SELECT * FROM admin WHERE admin_id = %s", (admin_id,))
    admin = cursor.fetchone()

    cursor.close()
    conn.close()

    return render_template('edit_admin_profile.html', admin=admin)



@app.route('/change-admin-password', methods=['POST'])
def change_admin_password():
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        return jsonify({'status': 'error', 'message': 'Please log in to change your password.'}), 401

    admin_id = session['admin_id']  # Get the logged-in admin's ID from the session

    # Get form data
    current_password = request.form['current_password']
    new_password = request.form['new_password']
    confirm_password = request.form['confirm_password']

    # Validate new password
    if new_password != confirm_password:
        return jsonify({'status': 'error', 'message': 'New passwords do not match.'}), 400

    if not re.match(r'^(?=.*[a-zA-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,}$', new_password):
        return jsonify({'status': 'error', 'message': 'Password must be at least 6 characters, contain letters, numbers, and symbols.'}),
400

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Verify current password
    cursor.execute("SELECT password FROM admin WHERE admin_id = %s", (admin_id,))
    admin = cursor.fetchone()

    if not admin or admin['password'] != current_password:
        return jsonify({'status': 'error', 'message': 'Current password is incorrect.'}), 400

    # Update the password
    cursor.execute("UPDATE admin SET password = %s WHERE admin_id = %s", (new_password, admin_id))
    conn.commit()
```

```
    cursor.close()
    conn.close()

    return jsonify({'status': 'success', 'message': 'Password updated successfully.'})



@app.route('/admin-dashboard')
def admin_dashboard():
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        flash('Please log in to access the dashboard.', 'danger')
        return redirect(url_for('login'))

    admin_id = session['admin_id']  # Get the logged-in admin's ID from the session

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Fetch admin details
    cursor.execute("SELECT name, profile_photo FROM admin WHERE admin_id = %s", (admin_id,))
    admin = cursor.fetchone()

    # Fetch dashboard statistics
    cursor.execute("SELECT COUNT(*) AS total_students FROM students")
    total_students = cursor.fetchone()['total_students']

    cursor.execute("SELECT COUNT(*) AS total_subjects FROM subjects")
    total_subjects = cursor.fetchone()['total_subjects']

    cursor.execute("SELECT COUNT(*) AS total_exams FROM exams")
    total_exams = cursor.fetchone()['total_exams']

    cursor.execute("SELECT MAX(generated_date) AS latest_mark FROM marksheet")
    latest_mark = cursor.fetchone()['latest_mark'] or "No marksheet available"

    cursor.close()
    conn.close()

    return render_template(
        "admin_dashboard.html",
        admin=admin,
        total_students=total_students,
        total_subjects=total_subjects,
        total_exams=total_exams,
        latest_mark=latest_mark
    )


@app.route('/student-profile')
def student_profile():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)  # Use dictionary=True for named columns
    cursor.execute("SELECT * FROM students ORDER BY id DESC")  # Fetch all students
    students = cursor.fetchall()  # Fetch all students from the database
    conn.close()
    students = sorted(students, key=lambda s: int(s['roll_no']))  # Sort students by roll number
    return render_template('student_profile.html', students=students)

import os
from flask import Flask, request, redirect, url_for
from werkzeug.utils import secure_filename


UPLOAD_FOLDER = 'static/images/student_photos'  # Directory to save uploaded files
ALLOWED_EXTENSIONS = {'csv', 'xlsx', 'png', 'jpg', 'jpeg', 'gif', 'webp'}

# app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.config['UPLOAD_FOLDER'] = os.path.join('static', 'images', 'student_photos')  # Ensure the path is correct

os.makedirs(UPLOAD_FOLDER, exist_ok=True)
```

```
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/add-student', methods=['POST'])
def add_student():
    name = request.form['name']
    student_class = request.form['student_class']  # input name in form must be 'student_class'
    dob = request.form['dob']
    father_name = request.form['father_name']
    mother_name = request.form['mother_name']
    registration_number = request.form['registration_number']
    roll_no = request.form['roll_no']
    email = request.form['email']
    password = request.form['password']

    profile_photo = request.files['profile_photo']
    filename = ''
    if profile_photo and allowed_file(profile_photo.filename):
        filename = secure_filename(profile_photo.filename)
        profile_photo.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO students (name, `class`, dob, profile_photo, father_name, mother_name, registration_number, roll_no, password,
email)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """, (name, student_class, dob, filename, father_name, mother_name, registration_number, roll_no, password, email))

    conn.commit()
    conn.close()

    flash('Student added successfully!', 'success')
    return redirect(url_for('student_profile'))

import pandas as pd  # Add this import for handling Excel/CSV files

@app.route('/bulk-upload-students', methods=['POST'])
def bulk_upload_students():
    if 'file' not in request.files:
        flash('No file part', 'danger')
        return redirect(url_for('student_profile'))

    file = request.files['file']

    if file.filename == '':
        flash('No selected file', 'danger')
        return redirect(url_for('student_profile'))

    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        try:
            # Read the file using pandas
            if filename.endswith('.csv'):
                data = pd.read_csv(file_path)
            elif filename.endswith('.xlsx'):
                data = pd.read_excel(file_path)
            else:
                flash('Unsupported file format. Please upload a CSV or Excel file.', 'danger')
                return redirect(url_for('student_profile'))

            # Iterate through the rows and insert into the database
            conn = get_db_connection()
            cursor = conn.cursor()

            for _, row in data.iterrows():
                cursor.execute("""
```

```
            INSERT INTO students (name, `class`, dob, profile_photo, father_name, mother_name, registration_number, roll_no,
password, email)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
          """, (
            row['name'],
            row['class'],
            row['dob'],
            row.get('profile_photo', ''),  # Optional field
            row['father_name'],
            row['mother_name'],
            row['registration_number'],
            row['roll_no'],
            row['password'],
            row['email']
          ))

        conn.commit()
        cursor.close()
        conn.close()

        flash('Students uploaded successfully!', 'success')
    except Exception as e:
        print("Error during bulk upload:", e)
        flash('An error occurred during bulk upload. Please check your file format.', 'danger')

    # Remove the uploaded file after processing
    os.remove(file_path)

  return redirect(url_for('student_profile'))

@app.route('/delete-student/<int:id>', methods=['POST'])
def delete_student(id):
  try:
    conn = get_db_connection()
    cursor = conn.cursor()

    # Delete the student from the database
    cursor.execute("DELETE FROM students WHERE id = %s", (id,))
    conn.commit()

    cursor.close()
    conn.close()

    flash('Student deleted successfully.', 'success')
  except Exception as e:
    print("Error deleting student:", e)
    flash('Failed to delete student. Please try again.', 'danger')

  return redirect(url_for('student_profile'))

@app.route('/edit-student/<int:id>', methods=['GET', 'POST'])
def edit_student(id):
  conn = get_db_connection()
  cursor = conn.cursor(dictionary=True)

  if request.method == 'POST':
    try:
      # Get updated details from the form
      name = request.form['name']
      student_class = request.form['student_class']
      roll_no = request.form['roll_no']
      registration_number = request.form['registration_number']
      father_name = request.form['father_name']
      mother_name = request.form['mother_name']
      dob = request.form['dob']
      email = request.form['email']
      password = request.form['password']

      # Check if a new profile photo is uploaded
      profile_photo = request.files.get('profile_photo')
      filename = None
      if profile_photo and allowed_file(profile_photo.filename):
```

```
            filename = secure_filename(profile_photo.filename)
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            profile_photo.save(file_path)
            print("File saved at:", file_path)

            # Update the student with the new photo
            cursor.execute("""
              UPDATE students
              SET name = %s, class = %s, roll_no = %s, registration_number = %s,
                father_name = %s, mother_name = %s, dob = %s, email = %s, profile_photo = %s
              WHERE id = %s
            """, (name, student_class, roll_no, registration_number, father_name, mother_name, dob, email, filename, id))
          else:
            # Update the student without changing the photo
            cursor.execute("""
              UPDATE students
              SET name = %s, class = %s, roll_no = %s, registration_number = %s,
                father_name = %s, mother_name = %s, dob = %s, email = %s, password = %s
              WHERE id = %s
            """, (name, student_class, roll_no, registration_number, father_name, mother_name, dob, email, password, id))

          conn.commit()
          flash('Student details updated successfully.', 'success')
        except Exception as e:
          print("Error during Edit Student:", e)
          flash("An error occurred while editing the student. Please try again.", "danger")
        finally:
          cursor.close()
          conn.close()

        return redirect(url_for('student_profile'))

    # Fetch the student details for the modal
    cursor.execute("SELECT * FROM students WHERE id = %s", (id,))
    student = cursor.fetchone()

    # Convert DOB to YYYY-MM-DD format
    if student and student['dob']:
      student['dob'] = student['dob'].strftime('%Y-%m-%d')

    cursor.close()
    conn.close()

    return jsonify(student)


@app.route('/subject-class', methods=['GET', 'POST'])
def subject_class_management():
    # Ensure the admin is logged in
    if 'admin_id' not in session:
      flash('Please log in to access this feature.', 'danger')
      return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    if request.method == 'POST':
      # Add a new subject or class
      name = request.form['name']
      code = request.form.get('code')  # Optional for classes
      type = request.form['type']  # 'subject' or 'class'

      if type == 'subject':
        cursor.execute("INSERT INTO subjects (name, code) VALUES (%s, %s)", (name, code))
      elif type == 'class':
        cursor.execute("INSERT INTO classes (name) VALUES (%s)", (name,))
      conn.commit()
      flash(f'{type.capitalize()} added successfully!', 'success')
      return redirect(url_for('subject_class_management'))

    # Fetch all subjects and classes
    cursor.execute("SELECT * FROM subjects")
```

```
    subjects = cursor.fetchall()

    cursor.execute("SELECT * FROM classes")
    classes = cursor.fetchall()

    cursor.close()
    conn.close()

    return render_template('subject_class_management.html', subjects=subjects, classes=classes)

@app.route('/edit-subject-class/<string:type>/<int:id>', methods=['POST'])
def edit_subject_class(type, id):
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        return jsonify({'status': 'error', 'message': 'Please log in to edit.'}), 401

    name = request.form['name']
    code = request.form.get('code')  # Optional for classes

    conn = get_db_connection()
    cursor = conn.cursor()

    if type == 'subject':
        cursor.execute("UPDATE subjects SET name = %s, code = %s WHERE id = %s", (name, code, id))
    elif type == 'class':
        cursor.execute("UPDATE classes SET name = %s WHERE id = %s", (name, id))
    conn.commit()

    cursor.close()
    conn.close()

    return jsonify({'status': 'success', 'message': f'{type.capitalize()} updated successfully!'})

@app.route('/delete-subject-class/<string:type>/<int:id>', methods=['POST'])
def delete_subject_class(type, id):
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        return jsonify({'status': 'error', 'message': 'Please log in to delete.'}), 401

    conn = get_db_connection()
    cursor = conn.cursor()

    if type == 'subject':
        cursor.execute("DELETE FROM subjects WHERE id = %s", (id,))
    elif type == 'class':
        cursor.execute("DELETE FROM classes WHERE id = %s", (id,))
    conn.commit()

    cursor.close()
    conn.close()

    return jsonify({'status': 'success', 'message': f'{type.capitalize()} deleted successfully!'})


@app.route('/student-detail/<int:student_id>', methods=['GET'])
def student_detail(student_id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Fetch student details
    cursor.execute("SELECT * FROM students WHERE id = %s", (student_id,))
    student = cursor.fetchone()

    if not student:
        flash("Student not found.", "danger")
        return redirect(url_for('student_profile'))

    # Fetch exam records for the student
    cursor.execute("""
        SELECT marks.id, exams.id AS exam_id, exams.name AS exam_name, subjects.id AS subject_id, subjects.name AS
subject_name, marks.marks_obtained
        FROM marks
```

```
    JOIN exams ON marks.exam_id = exams.id
    JOIN subjects ON marks.subject_id = subjects.id
    WHERE marks.student_id = %s
    ORDER BY exams.name, subjects.name
  """, (student_id,))
  records = cursor.fetchall()

  # Group records by exam
  exams_dict = {}
  for row in records:
    exam_id = row['exam_id']
    if exam_id not in exams_dict:
      exams_dict[exam_id] = {
        'exam_name': row['exam_name'],
        'subjects': []
      }
    exams_dict[exam_id]['subjects'].append(row)

  # Fetch all exams and subjects for the dropdowns
  cursor.execute("SELECT id, name FROM exams")
  exams = cursor.fetchall()
  cursor.execute("SELECT id, name FROM subjects")
  subjects = cursor.fetchall()

  # Calculate overall percentage
  cursor.execute("SELECT AVG(marks_obtained) AS percentage FROM marks WHERE student_id = %s", (student_id,))
  percentage = cursor.fetchone()['percentage'] or 0

  conn.close()

  return render_template(
    'student_dashboard_admin_side.html',
    student=student,
    exams_dict=exams_dict,
    percentage=round(percentage, 2),
    exams=exams,
    subjects=subjects
  )

@app.route('/generate-marksheet/<int:student_id>')
def generate_marksheet(student_id):
  exam_id = request.args.get('exam_id')
  conn = get_db_connection()
  cursor = conn.cursor(dictionary=True)

  cursor.execute("SELECT * FROM students WHERE id = %s", (student_id,))
  student = cursor.fetchone()

  if exam_id:
    cursor.execute("""
      SELECT exams.name AS exam_name,
          subjects.name AS subject_name,
          subjects.code AS subject_code,
          subjects.full_marks,
          marks.marks_obtained
      FROM marks
      JOIN exams ON marks.exam_id = exams.id
      JOIN subjects ON marks.subject_id = subjects.id
      WHERE marks.student_id = %s AND marks.exam_id = %s
    """, (student_id, exam_id))
  else:
    cursor.execute("""
      SELECT exams.name AS exam_name, subjects.name AS subject_name, marks.marks_obtained
      FROM marks
      JOIN exams ON marks.exam_id = exams.id
      JOIN subjects ON marks.subject_id = subjects.id
      WHERE marks.student_id = %s
    """, (student_id,))
  records = cursor.fetchall()
  conn.close()

  # qr_data = url_for('student_detail', student_id=student_id, _external=True)
```

```
    # img = qrcode.make(qr_data)
    # qr_path = f'static/qr_codes/qr_{student_id}.png'
    # img.save(qr_path)

    # Render HTML
    rendered = render_template('marksheet.html', student=student, records=records)

    path_wkhtmltopdf = r'C:\Users\sarbe\Downloads\wkhtmltox-0.12.6-1.mxe-cross-win64\wkhtmltox\bin\wkhtmltopdf.exe'  #
Update if your path is different
    config = pdfkit.configuration(wkhtmltopdf=path_wkhtmltopdf)

    pdf = pdfkit.from_string(rendered, False, configuration=config)


    # Send as download
    response = make_response(pdf)
    response.headers['Content-Type'] = 'application/pdf'
    response.headers['Content-Disposition'] = f'attachment; filename=marksheet_{student["name"]}.pdf'
    return response

# Secure Admin Access
@app.before_request
def require_login():
    if request.endpoint in ['marks_management', 'exam_marks_management', 'update_marks', 'update_all_marks'] and 'admin_id'
not in session:
        flash('Please log in to access this page.', 'danger')
        return redirect(url_for('login'))

# Marks Management Page
@app.route('/marks-management', methods=['GET'])
def marks_management():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Fetch all exams
    cursor.execute("SELECT * FROM exams ORDER BY id DESC")
    exams = cursor.fetchall()

    cursor.close()
    conn.close()

    return render_template('marks_management.html', exams=exams)

# Individual Exam Marks Management Page
@app.route('/exam-marks-management/<int:exam_id>', methods=['GET'])
def exam_marks_management(exam_id):
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Fetch exam details
    cursor.execute("SELECT * FROM exams WHERE id = %s", (exam_id,))
    exam = cursor.fetchone()

    # Fetch all students sorted by roll number
    cursor.execute("SELECT id, roll_no, registration_number, name FROM students ORDER BY roll_no ASC")
    students = cursor.fetchall()
    students = sorted(students, key=lambda s: int(s['roll_no']))

    # Fetch all subjects
    cursor.execute("SELECT id, name FROM subjects")
    subjects = cursor.fetchall()

    # Fetch existing marks for the exam
    cursor.execute("""
        SELECT marks.student_id, marks.subject_id, marks.marks_obtained, marks.remarks
        FROM marks
        WHERE marks.exam_id = %s
    """, (exam_id,))
    marks = cursor.fetchall()

    # Organize marks data for easy access
    marks_data = {}
```

```
    for mark in marks:
        marks_data[(mark['student_id'], mark['subject_id'])] = {
            'marks_obtained': mark['marks_obtained'],
            'remarks': mark['remarks']
        }

    cursor.close()
    conn.close()


    # Debugging output
    print("Subjects:", subjects)
    print("Marks Data:", marks_data)

    return render_template(
        'exam_marks_management.html',
        exam=exam,
        students=students,
        subjects=subjects,
        marks_data=marks_data
    )


# Update Marks for a Single Student
@app.route('/update-marks', methods=['POST'])
def update_marks():
    data = request.get_json()
    student_id = data['student_id']
    exam_id = data['exam_id']
    subject_marks = data['subject_marks']

    conn = get_db_connection()
    cursor = conn.cursor()

    for subject_mark in subject_marks:
        subject_id = subject_mark['subject_id']
        marks = subject_mark['marks']


        # Ensure marks is a valid integer and in range 0-100
        try:
            marks = int(marks)
        except (ValueError, TypeError):
            marks = 0
        if marks < 0 or marks > 100:
            marks = 0



        # Update or insert marks for the student
        cursor.execute("""
            INSERT INTO marks (student_id, exam_id, subject_id, marks_obtained)
            VALUES (%s, %s, %s, %s)
            ON DUPLICATE KEY UPDATE
            marks_obtained = VALUES(marks_obtained)
        """, (student_id, exam_id, subject_id, marks))

    conn.commit()
    cursor.close()
    conn.close()

    return jsonify({'message': 'Marks updated successfully!'})

# Update Marks for All Students
@app.route('/update-all-marks', methods=['POST'])
def update_all_marks():
    data = request.get_json()
    exam_id = data['exam_id']
    all_students_marks = data['all_students_marks']

    conn = get_db_connection()
    cursor = conn.cursor()
```

```
    for student_marks in all_students_marks:
        student_id = student_marks['student_id']
        for subject_mark in student_marks['subject_marks']:
            subject_id = subject_mark['subject_id']
            marks = subject_mark['marks']


                # Ensure marks is a valid integer and in range 0-100
            try:
                marks = int(marks)
            except (ValueError, TypeError):
                marks = 0
            if marks < 0 or marks > 100:
                marks = 0



            # Update or insert marks for the student
            cursor.execute("""
                INSERT INTO marks (student_id, exam_id, subject_id, marks_obtained)
                VALUES (%s, %s, %s, %s)
                ON DUPLICATE KEY UPDATE
                marks_obtained = VALUES(marks_obtained)
            """, (student_id, exam_id, subject_id, marks))

    conn.commit()
    cursor.close()
    conn.close()

    return jsonify({'status': 'success', 'message': 'All marks updated successfully!'})

@app.route('/add-edit-marks', methods=['POST'])
def add_edit_marks():
    student_id = request.form['student_id']
    exam_id = request.form['exam_id']
    subject_id = request.form['subject_id']
    marks = request.form['marks']

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO marks (student_id, exam_id, subject_id, marks_obtained)
        VALUES (%s, %s, %s, %s)
        ON DUPLICATE KEY UPDATE marks_obtained = %s
    """, (student_id, exam_id, subject_id, marks, marks))
    conn.commit()
    cursor.close()
    conn.close()
    flash('Marks saved successfully!', 'success')
    return redirect(url_for('student_detail', student_id=student_id))

@app.route('/exam-grade-setup', methods=['GET', 'POST'])
def exam_grade_setup():
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        flash('Please log in to access this feature.', 'danger')
        return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Fetch all exams
    cursor.execute("SELECT * FROM exams ORDER BY id DESC")
    exams = cursor.fetchall()

    # Fetch all grade thresholds
    cursor.execute("SELECT * FROM grades ORDER BY min_marks DESC")
    grades = cursor.fetchall()

    conn.close()
```

```
    return render_template(
        'exam_grade_setup.html',
        exams=exams,
        grades=grades
    )

@app.route('/add-edit-exam', methods=['POST'])
def add_edit_exam():
    exam_id = request.form.get('exam_id')
    name = request.form['name']

    conn = get_db_connection()
    cursor = conn.cursor()

    if exam_id:  # Edit exam
        cursor.execute("UPDATE exams SET name = %s WHERE id = %s", (name, exam_id))
    else:  # Add exam
        cursor.execute("INSERT INTO exams (name) VALUES (%s)", (name,))

    conn.commit()
    cursor.close()
    conn.close()

    flash('Exam saved successfully!', 'success')
    return redirect(url_for('exam_grade_setup'))

@app.route('/delete-exam/<int:exam_id>')
def delete_exam(exam_id):
    conn = get_db_connection()
    cursor = conn.cursor()
    # Delete rechecking requests for this exam
    cursor.execute("DELETE FROM rechecking_requests WHERE exam_id = %s", (exam_id,))
    # Delete marks for this exam
    cursor.execute("DELETE FROM marks WHERE exam_id = %s", (exam_id,))
    # Now delete the exam
    cursor.execute("DELETE FROM exams WHERE id = %s", (exam_id,))
    conn.commit()
    conn.close()
    flash('Exam and all related data deleted.', 'success')
    return redirect(url_for('exam_grade_setup'))

@app.route('/add-edit-grade', methods=['POST'])
def add_edit_grade():
    grade_id = request.form.get('grade_id')
    grade = request.form['grade']
    min_marks = request.form['min_marks']
    max_marks = request.form['max_marks']

    conn = get_db_connection()
    cursor = conn.cursor()

    if grade_id:  # Edit grade
        cursor.execute("""
            UPDATE grades
            SET grade = %s, min_marks = %s, max_marks = %s
            WHERE id = %s
        """, (grade, min_marks, max_marks, grade_id))
    else:  # Add grade
        cursor.execute("""
            INSERT INTO grades (grade, min_marks, max_marks)
            VALUES (%s, %s, %s)
        """, (grade, min_marks, max_marks))

    conn.commit()
    cursor.close()
    conn.close()

    flash('Grade saved successfully!', 'success')
    return redirect(url_for('exam_grade_setup'))

@app.route('/delete-grade/<int:grade_id>', methods=['GET'])
def delete_grade(grade_id):
```

```
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("DELETE FROM grades WHERE id = %s", (grade_id,))
    conn.commit()

    cursor.close()
    conn.close()

    flash('Grade deleted successfully!', 'success')
    return redirect(url_for('exam_grade_setup'))

@app.route('/notifications', methods=['GET', 'POST'])
def notifications():
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        flash('Please log in to access this feature.', 'danger')
        return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    if request.method == 'POST':
        # Add a new notification
        title = request.form['title']
        message = request.form['message']
        audience = request.form['audience']

        cursor.execute("""
            INSERT INTO notifications (title, message, audience)
            VALUES (%s, %s, %s)
        """, (title, message, audience))
        conn.commit()


        # Send email notifications if applicable
        if audience in ['all', 'students']:
            cursor.execute("SELECT email FROM students")
            recipients = [row['email'] for row in cursor.fetchall()]
        elif audience == 'admins':
            cursor.execute("SELECT email FROM admin")
            recipients = [row['email'] for row in cursor.fetchall()]

        if recipients:
            msg = Message(
                subject=f"New Notification: {title}",
                sender=app.config['MAIL_USERNAME'],
                recipients=recipients,
            )
            msg.html = f"""
            <!DOCTYPE html>
            <html>
            <head>
                <meta charset="UTF-8">
                <title>New Notification - Marksheet Portal</title>
                <style>
                    body {{
                        background: #f4f6fb;
                        font-family: 'Segoe UI', Arial, sans-serif;
                        margin: 0;
                        padding: 0;
                    }}
                    .email-container {{
                        max-width: 520px;
                        margin: 32px auto;
                        background: #fff;
                        border-radius: 14px;
                        box-shadow: 0 4px 24px rgba(44, 62, 80, 0.12);
                        padding: 32px 24px;
                        color: #222;
                    }}
                    .header {{
```

```
                text-align: center;
                margin-bottom: 18px;
            }}
            .header img {{
                width: 100px;
                margin-bottom: 8px;

            }}
            .notif-title {{
                color: #2d6cdf;
                font-size: 1.3em;
                font-weight: bold;
                margin-bottom: 10px;
            }}
            .notif-message {{
                background: #f0fdfa;
                border-left: 4px solid #2d6cdf;
                padding: 16px 18px;
                border-radius: 8px;
                margin-bottom: 18px;
                font-size: 1.08em;
            }}
            .details {{
                background: #f8fafc;
                border-radius: 8px;
                padding: 12px 18px;
                margin-bottom: 18px;
                font-size: 1em;
            }}
            .details strong {{
                color: #2d6cdf;
            }}
            .footer {{
                text-align: center;
                color: #888;
                font-size: 0.95em;
                margin-top: 24px;
            }}
            @media (max-width: 600px) {{
                .email-container {{ padding: 12px 4px; }}
            }}
        </style>
    </head>
    <body>
        <div class="email-container">
            <div class="header">
                <img src='https://png.pngtree.com/png-vector/20230107/ourmid/pngtree-new-update-banner-label-notification-
template-png-image_6554704.png' alt="Portal Logo">
                <h2>Marksheet Management Portal</h2>
            </div>
            <div class="notif-title">{title}</div>
            <div class="notif-message">{message}</div>
            <div class="details">
                <div><strong>Audience:</strong> {audience.capitalize()}</div>
                <div><strong>Date & Time:</strong> {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}</div>
            </div>
            <div class="footer">
                &copy; 2025 Marksheet Management Portal<br>
                <span style="color:#2d6cdf;">This is an automated message. Please do not reply.</span>
            </div>
        </div>
    </body>
    </html>
    """

    mail.send(msg)

flash('Notification added successfully!', 'success')
return redirect(url_for('notifications'))


# Handle search, filter, and pagination
```

```
    search_query = request.args.get('search', '')
    audience_filter = request.args.get('audience', 'all')
    page = int(request.args.get('page', 1))
    per_page = 5
    offset = (page - 1) * per_page

    query = "SELECT * FROM notifications WHERE 1=1"
    params = []

    if search_query:
        query += " AND title LIKE %s"
        params.append(f"%{search_query}%")

    if audience_filter != 'all':
        query += " AND audience = %s"
        params.append(audience_filter)

    query += " ORDER BY created_at DESC LIMIT %s OFFSET %s"
    params.extend([per_page, offset])

    cursor.execute(query, params)
    notifications = cursor.fetchall()

    # Get total count for pagination
    count_query = "SELECT COUNT(*) AS total FROM notifications WHERE 1=1"
    count_params = []

    if search_query:
        count_query += " AND title LIKE %s"
        count_params.append(f"%{search_query}%")

    if audience_filter != 'all':
        count_query += " AND audience = %s"
        count_params.append(audience_filter)

    cursor.execute(count_query, count_params)
    total_notifications = cursor.fetchone()['total']
    total_pages = (total_notifications + per_page - 1) // per_page

    cursor.close()
    conn.close()

    return render_template(
        'notifications.html',
        notifications=notifications,
        search_query=search_query,
        audience_filter=audience_filter,
        page=page,
        total_pages=total_pages
    )


@app.route('/delete-notification/<int:id>', methods=['POST'])
def delete_notification(id):
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        flash('Please log in to access this feature.', 'danger')
        return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("DELETE FROM notifications WHERE id = %s", (id,))
    conn.commit()

    cursor.close()
    conn.close()

    flash('Notification deleted successfully!', 'success')
    return redirect(url_for('notifications'))

@app.route('/student-notifications')
```

```
def student_notifications():
    # Ensure the student is logged in
    if 'student_id' not in session:
        flash('Please log in to access this feature.', 'danger')
        return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Fetch notifications for students or all audiences
    cursor.execute("""
        SELECT * FROM notifications
        WHERE audience = 'all' OR audience = 'students'
        ORDER BY created_at DESC
    """)
    notifications = cursor.fetchall()

    cursor.close()
    conn.close()

    return render_template('student_notifications.html', notifications=notifications)


@app.route('/settings', methods=['GET', 'POST'])
def settings():
    # Ensure the admin is logged in
    if 'admin_id' not in session:
        flash('Please log in to access this feature.', 'danger')
        return redirect(url_for('login'))

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    if request.method == 'POST':
        # Handle profile update
        if 'update_profile' in request.form:
            name = request.form['name']
            email = request.form['email']
            profile_photo = request.files['profile_photo']

            # Save profile photo if uploaded
            if profile_photo:
                photo_filename = f"admin_{session['admin_id']}.jpg"
                profile_photo.save(f"static/images/admin_photos/{photo_filename}")
                cursor.execute("UPDATE admin SET profile_photo = %s WHERE admin_id = %s", (photo_filename, session['admin_id']))

            # Update name and email
            cursor.execute("UPDATE admin SET name = %s, email = %s WHERE admin_id = %s", (name, email, session['admin_id']))
            conn.commit()
            flash('Profile updated successfully!', 'success')

        # Handle password change
        elif 'change_password' in request.form:
            current_password = request.form['current_password']
            new_password = request.form['new_password']
            confirm_password = request.form['confirm_password']

            # Verify current password
            cursor.execute("SELECT password FROM admin WHERE admin_id = %s", (session['admin_id'],))
            admin = cursor.fetchone()
            if admin['password'] != current_password:
                flash('Current password is incorrect!', 'danger')
            elif new_password != confirm_password:
                flash('New passwords do not match!', 'danger')
            else:
                # Update password
                cursor.execute("UPDATE admin SET password = %s WHERE admin_id = %s", (new_password, session['admin_id']))
                conn.commit()
                flash('Password changed successfully!', 'success')

    # Fetch admin details for the settings page
    cursor.execute("SELECT * FROM admin WHERE admin_id = %s", (session['admin_id'],))
```

```
    admin = cursor.fetchone()

    cursor.close()
    conn.close()

    return render_template('settings.html', admin=admin)

@app.route('/forgot_password', methods=['GET', 'POST'])
def forgot_password():
    if request.method == 'POST':
        username = request.form.get('username')
        user_type = request.form.get('role')  # 'student' or 'admin'

        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        if user_type == 'student':
            cursor.execute("SELECT * FROM students WHERE reg_no = %s", (username,))
        elif user_type == 'admin':
            cursor.execute("SELECT * FROM admin WHERE admin_id = %s", (username,))
        else:
            flash("Invalid role", "danger")
            return redirect(url_for('forgot_password'))

        user = cursor.fetchone()
        cursor.close()
        conn.close()

        if user:
            otp = str(random.randint(100000, 999999))
            session['reset_otp'] = otp
            session['reset_username'] = username
            session['reset_user_type'] = user_type
            session['reset_email'] = user['email']

            try:
                msg = Message('OTP for Password Reset', sender=app.config['MAIL_USERNAME'], recipients=[user['email']])
                msg.body = f"Your OTP for resetting your password is: {otp}"
                mail.send(msg)
                flash("OTP sent to your email.", "info")
                return redirect(url_for('verify_reset_otp'))
            except Exception as e:
                print("Email error:", e)
                flash("Failed to send OTP. Try again later.", "danger")
                return redirect(url_for('forgot_password'))
        else:
            flash("No account found for given username.", "danger")
            return redirect(url_for('forgot_password'))

    return render_template('forgot_password.html')


@app.route('/reset_password', methods=['GET', 'POST'])
def reset_password():
    if request.method == 'POST':
        new_password = request.form.get('new_password')
        confirm_password = request.form.get('confirm_password')

        if new_password != confirm_password:
            flash("Passwords do not match.", "danger")
            return redirect(url_for('reset_password'))

        if not re.match(r'^(?=.*[a-zA-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,}$', new_password):
            flash("Password must be at least 6 characters, contain letters, numbers, and symbols.", "danger")
            return redirect(url_for('reset_password'))

        username = session.get('reset_username')
        user_type = session.get('reset_user_type')

        conn = get_db_connection()
        cursor = conn.cursor()
```

```
        if user_type == 'student':
            cursor.execute("UPDATE students SET password = %s WHERE reg_no = %s", (new_password, username))
        elif user_type == 'admin':
            cursor.execute("UPDATE admin SET password = %s WHERE admin_id = %s", (new_password, username))

        conn.commit()
        cursor.close()
        conn.close()

        flash("Password updated successfully. You can now login!", "success")
        return redirect(url_for('login'))

    return render_template('reset_password.html')


@app.route('/new-password', methods=['GET', 'POST'])
def create_new_password():
    if request.method == 'POST':
        new_password = request.form.get('new_password')
        username = session.get('reset_user')
        user_type = session.get('reset_type')

        conn = get_db_connection()
        cursor = conn.cursor()

        if user_type == 'student':
            cursor.execute("UPDATE students SET password = %s WHERE reg_no = %s", (new_password, username))
        else:
            cursor.execute("UPDATE admin SET password = %s WHERE admin_id = %s", (new_password, username))

        conn.commit()
        cursor.close()
        conn.close()

        flash('Password updated successfully. You can now log in.', 'success')
        return redirect(url_for('login'))

    return render_template('new_password.html')

# ---------- BASIC PAGES ----------
@app.route('/')
def index():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    # Get the latest exam
    cursor.execute("SELECT id FROM exams ORDER BY id DESC LIMIT 1")
    latest_exam = cursor.fetchone()
    top_students = []
    if latest_exam:
        exam_id = latest_exam['id']
        # Get top 3 students for the latest exam by percentage
        cursor.execute("""
            SELECT s.name, s.roll_no, s.profile_photo,
                ROUND(SUM(m.marks_obtained) / SUM(sub.full_marks) * 100, 2) AS percentage
            FROM marks m
            JOIN students s ON m.student_id = s.id
            JOIN subjects sub ON m.subject_id = sub.id
            WHERE m.exam_id = %s
            GROUP BY m.student_id
            ORDER BY percentage DESC
            LIMIT 3
        """, (exam_id,))
        top_students = cursor.fetchall()


        # Fetch latest 5 notifications for all or students
    cursor.execute("""
        SELECT title, message, created_at
        FROM notifications
        WHERE audience = 'all' OR audience = 'admins'
        ORDER BY created_at DESC
```

```
        LIMIT 5
    """)
    latest_notices = cursor.fetchall()

    conn.close()
    return render_template('index.html', top_students=top_students)

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/services')
def services():
    return render_template('services.html')

@app.route('/gallery')
def gallery():
    return render_template('gallery.html')

@app.route('/features')
def features():
    return render_template('features.html')

@app.route('/privacy-policy')
def privacy_policy():
    return render_template('privacy_policy.html')

@app.route('/terms-and-conditions')
def terms_and_conditions():
    return render_template('terms_and_conditions.html')

@app.route('/help')
def help_page():
    return render_template('help.html')



# ---------- LOGOUT ----------
@app.route('/logout')
def logout():
    session.clear()
    flash('Logged out successfully.', 'success')
    return redirect(url_for('login'))



@app.errorhandler(404)
def page_not_found(e):
    return render_template('404.html'), 404

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000, debug=False)
```

An Autonomous Institution Affiliated to BPUT

Subject Specific Project Report

DOCUMENTATION

Step-by-Step Guide to Run the Marksheet Management Portal Application

Step-1: Install Python (version 3.8 or above) on your system if not already installed.

Step-2: Download or clone the project folder to your computer.

Step-3: Open the project folder in Visual Studio Code (VS Code).

Step-4: Open a new terminal in VS Code and create a virtual environment using the command python -m venv .venv

Step-5: Activate the virtual environment:

- On Windows: activate
- On Mac/Linux: source .venv/bin/activate

Step-6: Install all required dependencies using the command:

pip install -r requirements.txt

Step-7: Make sure MySQL server is running and the database is set up as per the provided SQL file.

Step-8: Open the app.py file in VS Code.

Step-9: Update the database connection details in app.py if needed (host, user, password, database).

Step-10: Run the application using the command:

python app.py

Step-11: Wait for the message indicating the Flask server is running (e.g., "Running on http://0.0.0.0:5000/").

Step-12: Open your web browser and go to http://localhost:5000 to access the Marksheet Management Portal.

Step-13: To add a student, log in as admin and use the "Add Student" option.

- Enter student details such as name, registration number, class, etc., and submit the form.

Step-14: To search for a student, use the search option and enter the registration number.

Step-15: To update student details, click on the "Edit" button next to the student's record, make changes, and save.

Step-16: To delete a student, click the "Delete" button next to the student's record and confirm deletion.

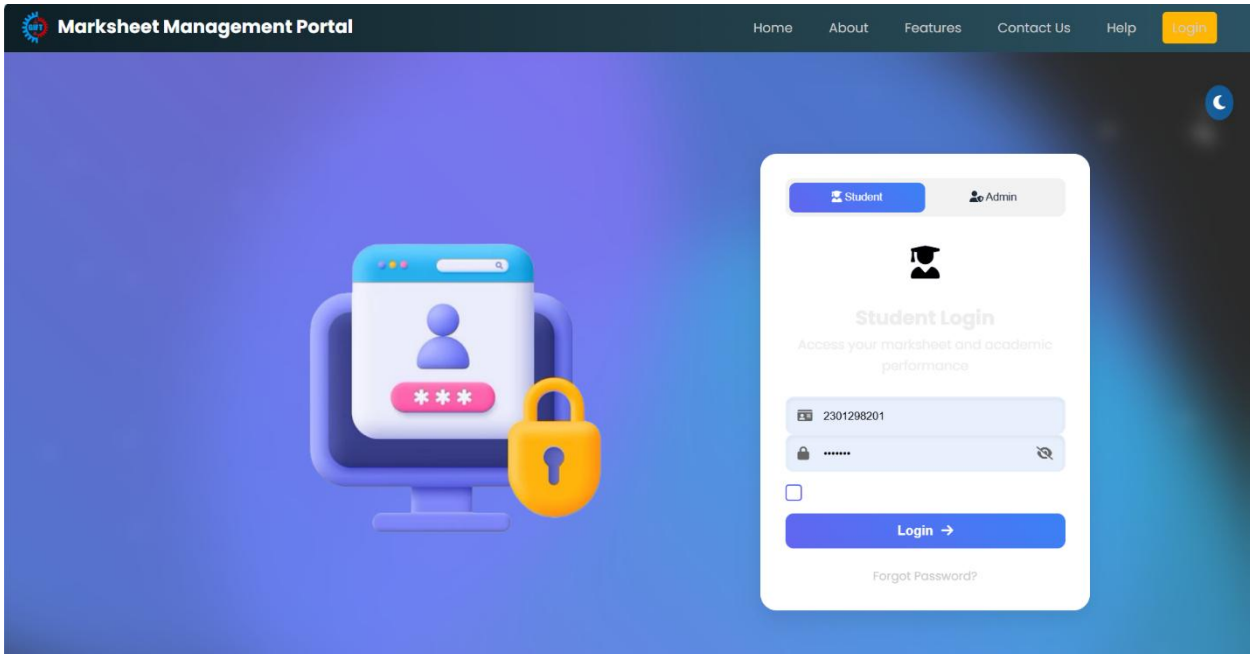
Step-17: To generate a marksheet, go to the student's profile and click the "Generate Marksheet" button.

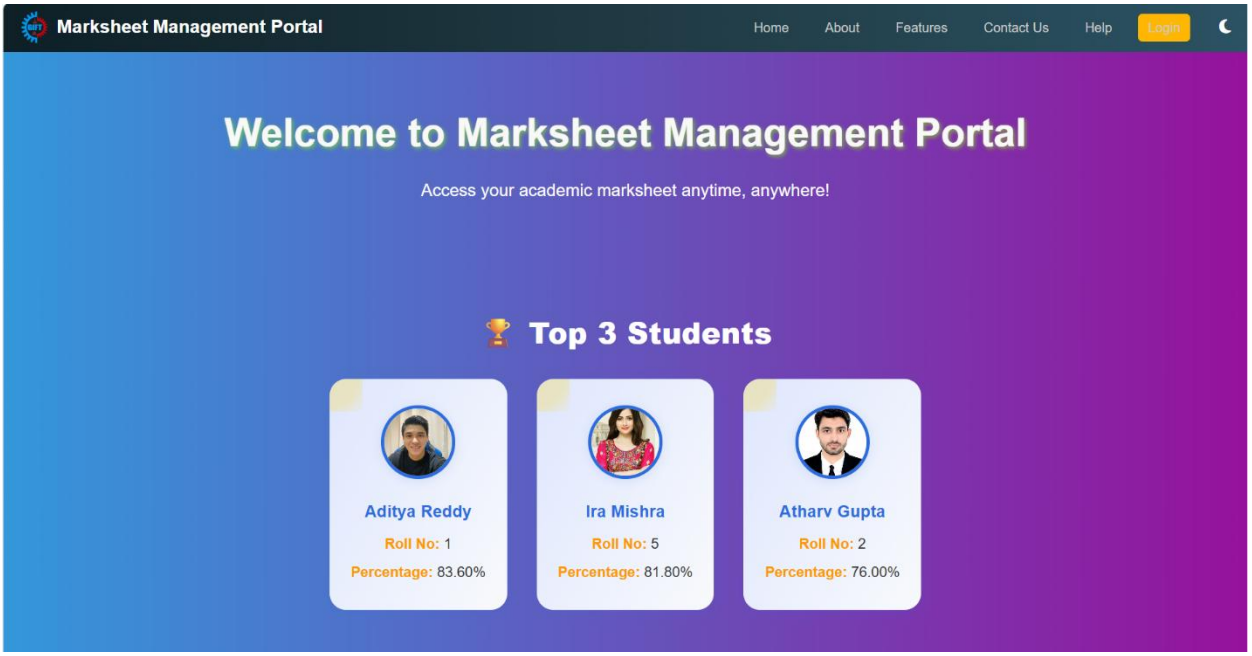
Step-18: For password recovery, use the "Forgot Password" link on the login page and follow the OTP verification process.

Step-19: For logout, click the "Logout" option in the navigation bar.

Step-20: When finished, stop the Flask server by pressing Ctrl+C in the terminal and close VS Code.

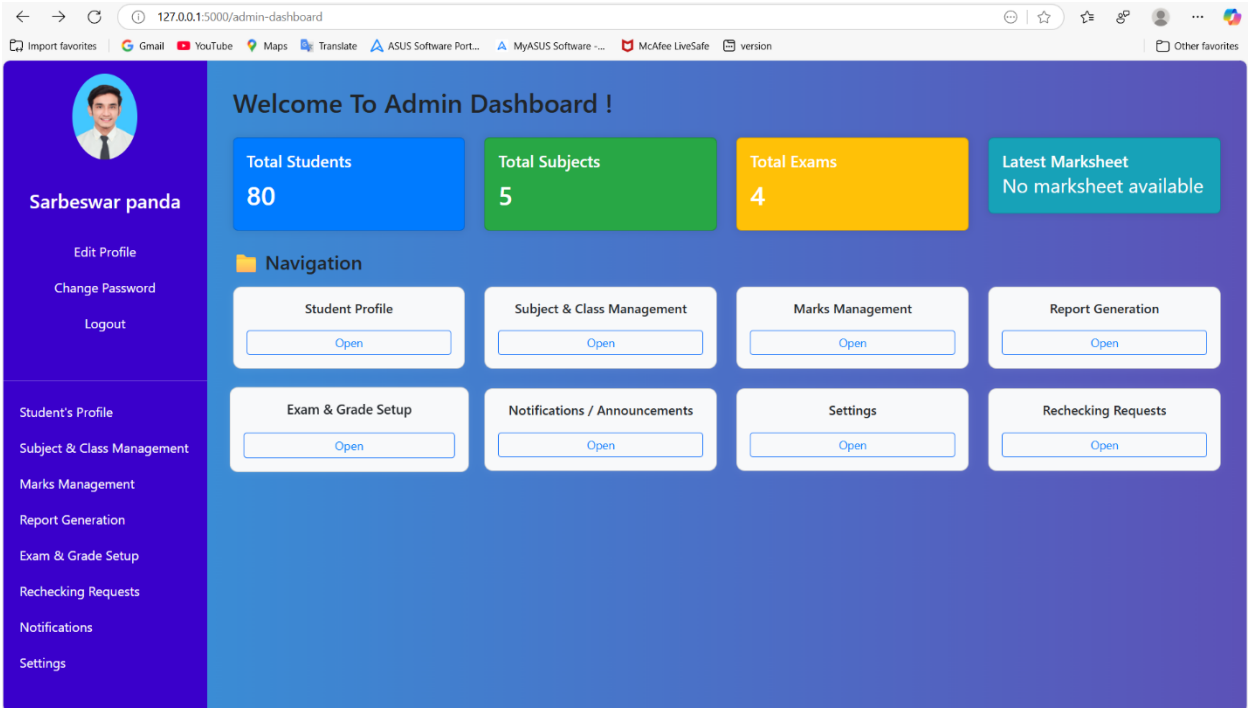
OUTPUT

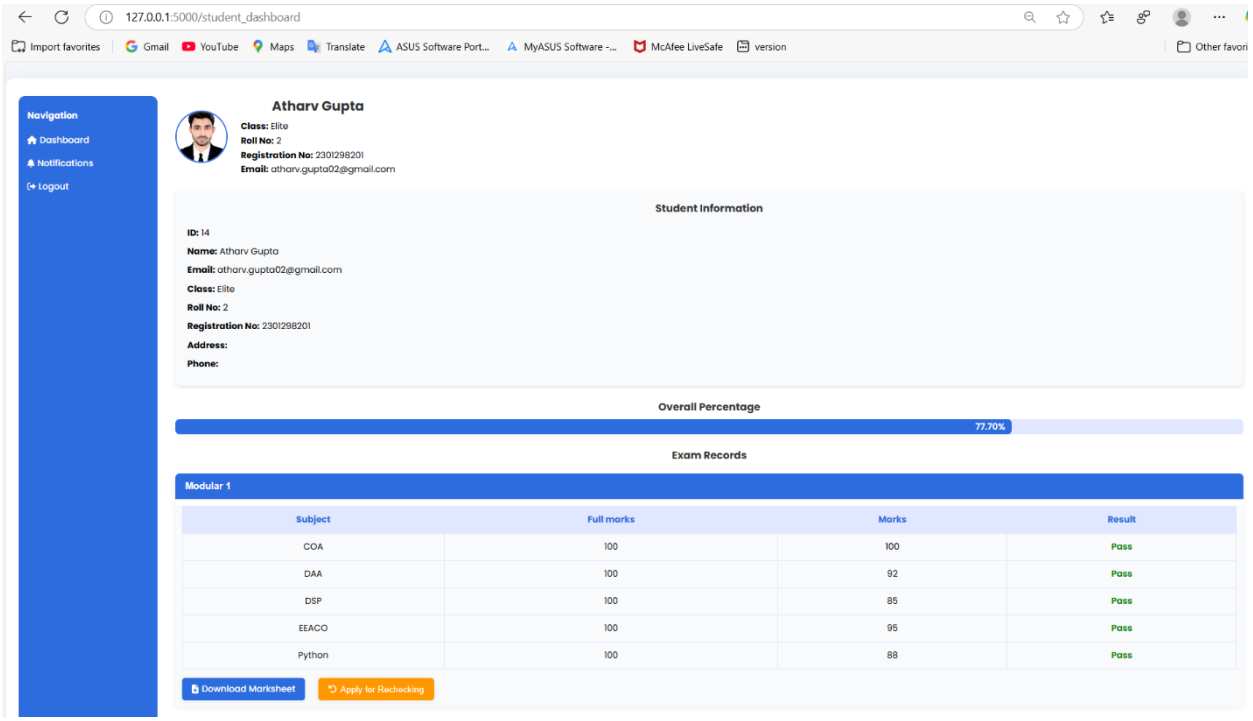

Marksheet Management Portal
Home
About
Features
Contact Us
Help
Login
Student
Admin
Student Login
Access your marksheet and academic performance
2301298201
Login
Forgot Password?


Marksheet Management Portal
Home
About
Features
Contact Us
Help
Login
Welcome to Marksheet Management Portal
Access your academic marksheet anytime, anywhere!
Top 3 Students
Aditya Reddy
Roll No: 1
Percentage: 83.60%
Ira Mishra
Roll No: 5
Percentage: 81.80%
Atharv Gupta
Roll No: 2
Percentage: 76.00%

An Autonomous Institution Affiliated to BPUT

Subject Specific Project Report





FUTURE ENHANCEMENT

The Marksheet Management Portal is designed with scalability and adaptability in mind. Several enhancements can be implemented in future versions to further improve functionality, user experience, and institutional value:

- **SMS/Email Notifications:** Integrate automated SMS and email alerts for result publication, announcements, and password recovery to keep users promptly informed.
- **Advanced Analytics:** Provide dashboards with graphical analysis of student performance, subject trends, and institutional statistics for both students and administrators.
- **Multi-language Support:** Add support for multiple languages to make the portal accessible to a wider and more diverse user base.
- **Mobile Application:** Develop dedicated Android and iOS apps for seamless access and push notifications on mobile devices.
- **Bulk Data Import/Export:** Enable bulk upload and download of student records, marks, and results using Excel or CSV files for easier data management.
- **Role-based Access Expansion:** Introduce additional roles such as faculty, parents, or department heads with customized permissions and dashboards.
- **Online Fee Payment Integration:** Allow students to pay academic fees online and generate digital receipts through the portal.

Document Verification: Implement digital signature and QR code verification for marksheets to enhance authenticity and prevent tampering

CONCLUSION

The Marksheet Management Portal successfully addresses the challenges associated with traditional marksheet management by providing a secure, efficient, and user-friendly digital solution for educational institutions. Through its robust features—including secure login, instant access to academic records, downloadable marksheets, real-time notifications, and a responsive interface—the portal enhances transparency and streamlines administrative workflows for students, faculty, and administrators alike. The integration of modules such as password recovery, FAQ/help, and contact support further improves the user experience and accessibility. Developed using modern web technologies and deployed on a scalable platform, the portal demonstrates practical full-stack development skills and serves as a reliable foundation for future enhancements. Overall, this project contributes significantly to the digital transformation of academic record management, ensuring accuracy, security, and convenience for all stakeholders.

REFERENCE

1. Flask Documentation: https://flask.palletsprojects.com/
2. MySQL Documentation: https://dev.mysql.com/doc/
3. SQLite Documentation: https://www.sqlite.org/docs.html
4. Jinja2 Templating: https://jinja.palletsprojects.com/
5. W3Schools HTML/CSS/JS Reference: https://www.w3schools.com/
6. MDN Web Docs (HTML, CSS, JavaScript): https://developer.mozilla.org/
7. SweetAlert2 Documentation: https://sweetalert2.github.io/
8. Render Deployment Docs: https://render.com/docs/deploy-flask
9. Stack Overflow: https://stackoverflow.com/
10. Bootstrap Documentation (for UI inspiration): https://getbootstrap.com/
11. Font Awesome Icons: https://fontawesome.com/
12. Python Official Documentation: https://docs.python.org/3/

GitHub (for open-source project structure inspiration): https://github.com/

THANK YOU