Prompt Log

Asma Haider	Uzair Abdullah
BSSE23051	BSSE23075

okay

let's just add edit button and files to comply it too

---

and yes

you talked about removing the image file name from csv too

should we remove it rn

why read from csv?

we should use db na?

---

look

this csv file is for adding bulk data

main data place is sqlite

---

you my friend

why you acting like uou dont know the context?

you're mentioneing wrong things

you know somethings and acting like u dont know

table is already there

we can enrolll single student too}

and we can add bulk too

db is yhere tables are there

csv uploading option is there

tell me where were we?

i think u forgot all the above chat

---

tell me which files hosuld i paste here

so the work is consistent

---

the students are shown in students.html not index.html

---

app.py:

```
import os
from werkzeug.utils import secure_filename
from flask import Flask, render_template, redirect, url_for, flash, request
from models import db, Student, Room, Expense, Issue, Admin, FeeRecord
from datetime import datetime
from forms import EnrollForm, ExpenseForm, IssueForm,
AdminLoginForm,AdminRegisterForm , FeeCollectionForm
from flask_migrate import Migrate
from flask_login import LoginManager, login_user, login_required, logout_user,
current_user
from flask_bcrypt import Bcrypt
from flask import send_file
from io import BytesIO
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from calendar import month_name
from sqlalchemy import extract
import pandas as pd
import openpyxl
from flask_wtf.csrf import CSRFProtect, validate_csrf, CSRFError
```

tell me

i wanna simplify the work

i havnt used sqlchmay and things before

what if we remove the picture thing for now

and add later maybe?

---

when i pressed the dleye button

the screen got kinda dim and wont let me click any button

---

no im saying like if we remove image of students thing totally from the project and add later while hsifting to the aws (using cloud 9 maybe)

what do you say

will it be easy?

---

the pictures are being loaded in many pages

enroll.html contains pic uploading

tell the changes for image uploadation removing for now

---

when i added a student to a room which was already full

it still let me add and look

even when i've logged in once

on accessing any page

---

still asks to log in

can't it happen ke when the admin logs in

he stays logged in and can access any page until logs out again

---

I noticed ke the project mentions and have issues.html tooo

but not shown in ui

what steps should i follow

lets add a dropdown for status

like:

pending

etc instead of wroting status by hand

---

okay

tell me the steps to run the project one lsat time

i've pushed the code iwth venv folder to github and my freind will get it from there

how should she run it

---

it was a flask app

ypuo forgot?

---

i'm giving them the whole folder naaa

---

why doing these things

they'll just run the app

use the same db which im using

---

so why not

flask run?

---

okay now let's do this step by step

since you know we have the project on pc and we have to shift and host it on aws

what steps should be followed?

tell (about the setup) comprehensively about user data

ec2s,s3s,ecs,codecommit etc., rds security groups and all other tools mentioned in the document

Full guide for setting up and shifting the app to aws

---

no

not like this

---

you've to be comprehensive

give every single step

if i've to create a file or copy a file show the folder structure every time

---

go step by step (give one module then ask then go)

become an assistant who knows everything and treat me as a noob

---

should i give u my current folder structure

so u can know ke which file's where

---

let's start again

---

and in issues.html

this is the view

i think there should be buttons for

if i've resolved the issue then i can change the status too

and the button for adding the issue is dull

---

when i added an expense:

this error in flash came:

Please check the form inputs.

---

and the deletion issue is still there:

Internal Server Error

The server encountered an internal error and was unable to complete your request.
Either the server is overloaded or there is an error in the applicati

i'll check the logs and paste those too:

---

can't i turn off or remove this csrf token completely?

---

2. Security Keys (Hardcoded for stability in this lab)

app.config['SECRET_KEY'] = 'hostel-master-key-secure-123'

app.config['WTF_CSRF_SECRET_KEY'] = 'hostel-csrf-key-secure-123'

3. Session Settings (Required for HTTP Load Balancer)

app.config['WTF_CSRF_ENABLED'] = False

already disabled

---

and one more thing

i wanna remove the picture field from sample excel file

so the admin upload the pictures later by editing the student

but i wanna make sure that there's no conflict in excel uploadation if i remove the field totally

---

look for places where the excel bulk upload implementation is:

---

on deleting a student this error came:

Error deleting student: 'Room' object has no attribute 'current_occupancy'

i'll copy paste the related files:

---

and one more thing

when i go from rooms page to add a student

it should keep the room no already there in the form

but we still have to put the room again:

---

what about checking the post method?

---

on deleting still again:

Error deleting student: Entity namespace for "issue" has no property "student_id"

---

and in dashboard

there's analytics of:

fully paid partially paid and unpaid

there's a fully paid

but not showing here

---

the student with fully paid didn't show up in dashboard even after this

---

okay listen

instead of fixing let's remove that(fully/partially/un paid) analytics from dashboard

---

and let's add new analytics

add many as u can

little from little

our prof said

there should be more analytics on dashboard

previous dashboard:

okay let's add ses now

You do not have sufficient access to perform this action.

User: arn:aws:sts::891377334911:assumed-role/voclabs/user4452789=bsse23051@itu.edu.pk is not authorized to perform: ses:GetAccount on resource: * because no identity-based policy allows the ses:GetAccount action

---

no

my friend did add this service

okay ?

i wanna add it too

---

we should add it on login

or registration ?

---

let's do it step by step then

first tell how can i setup my account for this

then go for code

---

can't we set time for otp to expire?

---

give the changes step by step

---

the flash shown on otp screen is so light that it was so hard to read

---

and there should be a timer of 5 mins

and after that the verify button should not work

and can the structure of email being sent be more beautiful and professional?

---

i'm putting the right otp

but the page is still there

it just refreshes and nothing happens

---

and i caught a big problem

the sidebar is shown on the log in page too

it should not be there

otherwise the un signed in admin can access anything

---

and i caught a big problem

the sidebar is shown on the log in page and otp page too

it should not be there

otherwise the un signed in admin can access anything

---

and the other problem is that

when i put otp the page reloads and the input field with otp icon just move up a bit and nothing happens

---

sidebar still showing on login signup and otp pages

---

look at the ss

the flash msg is too light to read

and the styling from the form is gone

---

nav bar still here

what if we remove them manually

and the flash cards shown by flash() are too light in colors

can't be read

---

look at the ss

there are double flash msgs showing

and the side bar at login and signup page is still present

---

app = Flask(__name__)

app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY', 'your_secret_key')

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///hostel.db'

app.config['UPLOAD_FOLDER'] = 'static/uploads'

app.config['PERMANENT_SESSION_LIFETIME'] = 1800 # 30 minutes

```
app.config['WTF_CSRF_ENABLED'] = True
app.config['WTF_CSRF_SECRET_KEY'] = os.environ.get('WTF_CSRF_SECRET_KEY', 'your_csrf_secret_key')

# Initialize CSRF protection
csrf = CSRFProtect(app)

db.init_app(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
login_manager.login_view = 'admin_login'

# After initializing db
migrate = Migrate(app, db)

def create_tables():
    db.create_all()
    for i in range(1, 9):
        if not Room.query.filter_by(room_number=i).first():
            room = Room(room_number=i)
            db.session.add(room)
    db.session.commit()

@login_manager.user_loader
def load_user(user_id):
    return Admin.query.get(int(user_id))

@app.route('/')
```

```
def home():
    return render_template('Home.html')


# Admin Login
@app.route('/admin_login', methods=['GET', 'POST'])
def admin_login():
    form = AdminLoginForm()
    if form.validate_on_submit():
        admin = Admin.query.filter_by(username=form.username.data).first()
        if admin and bcrypt.check_password_hash(admin.password_hash,
form.password.data):
            login_user(admin)
            return redirect(url_for('admin_dashboard'))
        else:
            flash('Login Unsuccessful. Please check username and password', 'danger')
    return render_template('admin_login.html', form=form)


# Admin Dashboard
@app.route('/admin_dashboard')
@login_required
def admin_dashboard():
    # Get current year and month
    current_year = datetime.now().year
    current_month = datetime.now().month


    # Get total number of active students
    total_students = Student.query.filter_by(status='active').count()
```

```
# Get monthly expenses for the last 6 months
monthly_expenses = []
monthly_income = []
months = []

for i in range(5, -1, -1):
    month = current_month - i
    year = current_year
    if month <= 0:
        month += 12
        year -= 1

    # Get expenses for the month
    month_expenses = Expense.query.filter(
        db.extract('year', Expense.date) == year,
        db.extract('month', Expense.date) == month
    ).all()
    total_expense = sum(expense.price for expense in month_expenses)

    # Get income (fee collections) for the month
    month_income = FeeRecord.query.filter(
        db.extract('year', FeeRecord.date_paid) == year,
        db.extract('month', FeeRecord.date_paid) == month
    ).all()
    total_income = sum(record.amount for record in month_income)

    monthly_expenses.append(total_expense)
    monthly_income.append(total_income)
```

```
    months.append(month_name[month][:3])  # Short month name

# Get expense categories for pie chart
expense_categories = db.session.query(
    Expense.item_name,
    db.func.sum(Expense.price).label('total')
).group_by(Expense.item_name).all()

# Calculate total expenses and income for the current month
current_month_expenses = sum(monthly_expenses[-1:])
current_month_income = sum(monthly_income[-1:])

# Calculate profit/loss
profit_loss = current_month_income - current_month_expenses

# Get fee collection status
fully_paid = Student.query.filter_by(status='active').filter(
    Student.fee_status == 'paid'
).count()
partially_paid = Student.query.filter_by(status='active').filter(
    Student.fee_status == 'partial'
).count()
unpaid = Student.query.filter_by(status='active').filter(
    Student.fee_status == 'unpaid'
).count()

return render_template('admin_dashboard.html',
                       total_students=total_students,
```

```
                monthly_expenses=monthly_expenses,
                monthly_income=monthly_income,
                months=months,
                expense_categories=expense_categories,
                current_month_expenses=current_month_expenses,
                current_month_income=current_month_income,
                profit_loss=profit_loss,
                fully_paid=fully_paid,
                partially_paid=partially_paid,
                unpaid=unpaid)

# Room Management (Admin Only)
@app.route('/room_management')
@login_required
def room_management():
    return render_template('room_management.html')

# Add this function at the beginning of app.py
def allowed_file(filename):
    ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif', 'xlsx', 'xls'}
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/enroll', methods=['GET', 'POST'])
@login_required
def enroll():
    form = EnrollForm()
    if form.validate_on_submit():
        # Check if the file is present and allowed
```

```
if 'picture' not in request.files:
    flash('No picture file provided', 'danger')
    return redirect(request.url)

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

    # Create a new student object
    student = Student(
        name=form.name.data,
        fee=form.fee.data,
        room_id=form.room_number.data,
        picture=filename  # Store filename in the database
    )

    # Add student to the database
    try:
        db.session.add(student)
        db.session.commit()
        flash('Student enrolled successfully!', 'success')
        return redirect(url_for('students'))
    except Exception as e:
        db.session.rollback()
        flash(f'Error enrolling student: {e}', 'danger')
else:
```

```
            flash('Invalid file format. Allowed types are png, jpg, jpeg, gif.', 'danger')

    return render_template('enroll.html', form=form)

@app.template_filter('month_name')
def month_name_filter(month_number):
    return month_name[month_number]



# from calendar import month_name

@app.route('/expenses', methods=['GET', 'POST'])
@login_required
def expenses():
    form = ExpenseForm()

    # Get filter parameters (default to current month and year)
    month = request.args.get('month', datetime.now().month, type=int)
    year = request.args.get('year', datetime.now().year, type=int)

    # Get the previous month's data
    prev_month = month - 1 if month > 1 else 12
    prev_year = year if month > 1 else year - 1

    # Query expenses for the current month and year
    expenses_current = Expense.query.filter(
        extract('year', Expense.date) == year,
        extract('month', Expense.date) == month
```

```
).order_by(Expense.date.desc()).all()

# Query expenses for the previous month and year
expenses_previous = Expense.query.filter(
    extract('year', Expense.date) == prev_year,
    extract('month', Expense.date) == prev_month
).order_by(Expense.date.desc()).all()

# Calculate totals for both months
total_expenses_current = sum(expense.price for expense in expenses_current)
total_expenses_previous = sum(expense.price for expense in expenses_previous)

# Get fee collections for current month
fee_records_current = FeeRecord.query.filter(
    extract('year', FeeRecord.date_paid) == year,
    extract('month', FeeRecord.date_paid) == month
).all()
total_income_current = sum(record.amount for record in fee_records_current)

# Get fee collections for previous month
fee_records_previous = FeeRecord.query.filter(
    extract('year', FeeRecord.date_paid) == prev_year,
    extract('month', FeeRecord.date_paid) == prev_month
).all()
total_income_previous = sum(record.amount for record in fee_records_previous)

# Calculate remaining balance
remaining_balance_current = total_income_current - total_expenses_current
```

```
remaining_balance_previous = total_income_previous - total_expenses_previous


if form.validate_on_submit():
    expense = Expense(
        item_name=form.item_name.data,
        price=form.price.data,
        date=form.date.data,
        user_id=current_user.id
    )
    db.session.add(expense)
    db.session.commit()
    flash('Expense added successfully!', 'success')
    return redirect(url_for('expenses'))


return render_template('expenses.html',
                       form=form,
                       expenses_current=expenses_current,
                       expenses_previous=expenses_previous,
                       total_expenses_current=total_expenses_current,
                       total_expenses_previous=total_expenses_previous,
                       total_income_current=total_income_current,
                       total_income_previous=total_income_previous,
                       remaining_balance_current=remaining_balance_current,
                       remaining_balance_previous=remaining_balance_previous,
                       current_month=month,
                       current_year=year,
                       prev_month=prev_month,
                       prev_year=prev_year,
```

```
                month_names=month_name)


@app.route('/export_pdf/<int:year>/<int:month>', methods=['GET'])
def export_pdf(year, month):
    # Get expenses for the specified month and year
    expenses = Expense.query.filter(
        db.extract('year', Expense.date) == year,
        db.extract('month', Expense.date) == month
    ).all()

    # Calculate total expenses for the month
    total_expenses = sum(expense.price for expense in expenses)

    # Get total income from fee records for the month
    total_income = db.session.query(db.func.sum(FeeRecord.amount)).filter(
        db.extract('year', FeeRecord.date_paid) == year,
        db.extract('month', FeeRecord.date_paid) == month
    ).scalar() or 0

    # Calculate remaining balance
    remaining_balance = total_income - total_expenses

    # Create a BytesIO buffer for the PDF
    buffer = BytesIO()
    pdf = canvas.Canvas(buffer, pagesize=A4)
    width, height = A4

    # Title section
```

```
pdf.setFont("Helvetica-Bold", 16)
pdf.drawString(200, height - 50, f"Monthly Expense Report - {month}/{year}")

# Income and Expense Summary
pdf.setFont("Helvetica-Bold", 12)
pdf.drawString(50, height - 100, f"Total Income: Rs {total_income}")
pdf.drawString(50, height - 120, f"Total Expenses: Rs {total_expenses}")
pdf.drawString(50, height - 140, f"Remaining Balance: Rs {remaining_balance}")

# Table headers
y_position = height - 180
pdf.setFont("Helvetica-Bold", 12)
pdf.drawString(50, y_position, "Item Name")
pdf.drawString(250, y_position, "Price (Rs)")
pdf.drawString(400, y_position, "Date")

# Table content
pdf.setFont("Helvetica", 10)
y_position -= 20
for expense in expenses:
    pdf.drawString(50, y_position, expense.item_name)
    pdf.drawString(250, y_position, f"Rs {expense.price}")
    pdf.drawString(400, y_position, expense.date.strftime('%Y-%m-%d'))
    y_position -= 20
    if y_position < 50:
        pdf.showPage()
        y_position = height - 50
```

```
    # Total summary section
    y_position -= 30
    pdf.setFont("Helvetica-Bold", 12)
    pdf.drawString(50, y_position, f"Total Expenses for {month}/{year}: Rs {total_expenses}")
    y_position -= 20
    pdf.setFont("Helvetica-Bold", 12)
    pdf.drawString(50, y_position, f"Remaining Balance: Rs {remaining_balance if remaining_balance >= 0 else 0}")

    # Footer
    pdf.setFont("Helvetica", 10)
    pdf.drawString(50, 30, "Report Generated By: Najam Ali")
    pdf.drawString(400, 30, "Report Approved By: Mr Bilal")

    pdf.save()
    buffer.sk(0)

    # Send the PDF as a file download
    return send_file(buffer, as_attachment=True, download_name=f"Expense_Report_{month}_{year}.pdf", mimetype='application/pdf')

# Edit expense route
@app.route('/edit_expense/<int:expense_id>', methods=['POST'])
def edit_expense(expense_id):
    expense = Expense.query.get_or_404(expense_id)
    expense.item_name = request.form['item_name']
    expense.price = float(request.form['price'])
    expense.date = request.form['date']
```

```
    db.session.commit()
    flash('Expense updated successfully!', 'success')
    return redirect(url_for('expenses', year=expense.date.year,
month=expense.date.month))

# Delete expense route
@app.route('/delete_expense/<int:expense_id>', methods=['POST'])
def delete_expense(expense_id):
    expense = Expense.query.get_or_404(expense_id)
    db.session.delete(expense)
    db.session.commit()
    flash('Expense deleted successfully!', 'success')
    return redirect(url_for('expenses'))

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



@app.route('/students')
@login_required
def students():
    page = request.args.get('page', 1, type=int)
    per_page = 10
    paginated_students = Student.query.paginate(page=page, per_page=per_page)
    form = EnrollForm()  # Create an instance of the form for CSRF token
    return render_template('students.html', students=paginated_students, form=form)
```

```
@app.route('/rooms')
@login_required
def rooms():
    all_rooms = Room.query.all()
    return render_template('rooms.html', rooms=all_rooms)

# Admin Registration

@app.route('/admin_register', methods=['GET', 'POST'])
def admin_register():
    form = AdminRegisterForm()
    if form.validate_on_submit():
        existing_admin = Admin.query.filter_by(email=form.email.data).first()
        if existing_admin:
            flash('Email already in use', 'danger')
        else:
            hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
            new_admin = Admin(name=form.name.data, username=form.username.data,
email=form.email.data, password_hash=hashed_password)
            try:
    if form.validate_on_submit():
        student_name = form.student_name.data
        amount = form.amount.data
        date_paid = form.date.data

        student = Student.query.filter_by(name=student_name).first()
```

```
        if student:
            # Check if the payment would exceed the total fee
            if amount > student.remaining_fee:
                flash(f'Payment amount exceeds remaining fee of Rs {student.remaining_fee}!',
'warning')
            else:
                fee_record = FeeRecord(
                    student_id=student.id,
                    amount=amount,
                    date_paid=date_paid
                )
                student.last_fee_payment = date_paid
                db.session.add(fee_record)
                db.session.commit()
                flash('Fee payment recorded successfully!', 'success')
                return redirect(url_for('collect_fee'))
        else:
            flash('Student not found', 'danger')

    return render_template('collect_fee.html',
                           form=form,
                           fee_records=fee_records,
                           total_fee=total_fee,
                           current_month=month,
                           current_year=year,
                           month_name=month_name,
                           fully_paid_students=fully_paid_students,
```

```
                partially_paid_students=partially_paid_students,
                unpaid_students=unpaid_students)


@app.template_filter('month_name')  # Register the filter
def month_name_filter(month_number):
    return month_name[month_number]

@app.route('/update_students_excel', methods=['GET', 'POST'])
@login_required
def update_students_excel():
    if request.method == 'POST':
        if 'excel_file' not in request.files:
            flash('No file uploaded', 'danger')
            return redirect(request.url)

        file = request.files['excel_file']
        if file and allowed_file(file.filename):
            try:
                # Read the Excel file
                print(f"Reading Excel file: {file.filename}")  # Debug log
                df = pd.read_excel(file)
                print(f"Excel data shape: {df.shape}")  # Debug log
                print(f"Excel columns: {df.columns.tolist()}")  # Debug log

                # Process each row
                for index, row in df.iterrows():
                    print(f"Processing row {index}: {row.to_dict()}")  # Debug log
```

```
            student = Student.query.filter_by(name=row['name']).first()

            if student:
                print(f"Updating existing student: {student.name}")  # Debug log
                # Update existing student
                student.fee = float(row['fee'])
                student.room_id = int(row['room_id'])
                student.status = row.get('status', 'active')

                # Update picture if provided
                if 'picture' in row and pd.notna(row['picture']):
                    picture_filename = secure_filename(row['picture'])
                    if os.path.exists(os.path.join(app.config['UPLOAD_FOLDER'],
picture_filename)):
                        student.picture = picture_filename
            else:
                print(f"Creating new student: {row['name']}")  # Debug log
                # Create new student
                new_student = Student(
                    name=row['name'],
                    fee=float(row['fee']),
                    room_id=int(row['room_id']),
    return render_template('update_students_excel.html')

@app.route('/download_sample_excel')
@login_required
def download_sample_excel():
    # Create a sample DataFrame
```

```
    df = pd.DataFrame({
        'name': ['John Doe', 'Jane Smith'],
        'fee': [5000, 6000],
        'room_id': [1, 2],
        'status': ['active', 'active'],
        'picture': ['student1.jpg', 'student2.jpg']  # Add picture field
    })

    # Create a BytesIO buffer
    buffer = BytesIO()

    # Write the DataFrame to Excel
    with pd.ExcelWriter(buffer, engine='openpyxl') as writer:
        df.to_excel(writer, index=False, sheet_name='Students')

    # Set the buffer position to the beginning
    buffer.seek(0)

    # Send the file
    return send_file(
        buffer,
        mimetype='application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet',
        as_attachment=True,
        download_name='sample_student_template.xlsx'
    )

@app.route('/about')
```

```
def about():
    current_year = datetime.now().year
    return render_template('about.html', current_year=current_year)

@app.route('/delete_student/<int:student_id>', methods=['POST'])
@login_required
def delete_student(student_id):
    # Only check CSRF token validity
    try:
        validate_csrf(request.form.get('csrf_token'))
    except CSRFError:
        flash('Invalid CSRF token. Please try again.', 'danger')
        return redirect(url_for('students'))

    try:
        student = Student.query.get_or_404(student_id)
        # Delete associated fee records first
        FeeRecord.query.filter_by(student_id=student_id).delete()
        # Delete the student's picture file if it exists
        if student.picture:
            picture_path = os.path.join(app.config['UPLOAD_FOLDER'], student.picture)
            if os.path.exists(picture_path):
                os.remove(picture_path)
        db.session.delete(student)
        db.session.commit()
        flash('Student deleted successfully!', 'success')
    except Exception as e:
        db.session.rollback()
```

```
        flash(f'Error deleting student: {str(e)}', 'danger')
    return redirect(url_for('students'))

if __name__ == '__main__':
    with app.app_context():
        create_tables()
    app.run(debug=True,port=5051)
```

students.html:

```
{% extends 'base.html' %}

{% block title %}Students{% endblock %}

{% block content %}
<h1 class="text-center">Students</h1>
<div class="d-flex justify-content-center mb-4">
    <a href="{{ url_for('update_students_excel') }}" class="btn btn-success">
        <i class="fas fa-file-excel"></i> Update via Excel
    </a>
</div>
<div class="d-flex justify-content-center">
    <div class="card w-75">
        <div class="card-body">
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Name</th>
```

```
            <th>Room</th>
            <th>Fee</th>
            <th>Picture</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {% for student in students.items %}
          <tr>
            <td>{{ student.name }}</td>
            <td>{{ student.room.room_number }}</td>
            <td>{{ student.fee }}</td>
            <td><img src="{{ url_for('static', filename='uploads/' ~ student.picture) }}" alt="Student Picture" width="100"></td>
            <td>
              <button type="button" class="btn btn-danger btn-sm" data-bs-toggle="modal" data-bs-target="#deleteModal{{ student.id }}">
                <i class="fas fa-trash"></i> Delete
              </button>
            </td>
          </tr>
          {% endfor %}
        </tbody>
      </table>

      <!-- Pagination Controls -->
      <nav aria-label="Page navigation">
        <ul class="pagination justify-content-center">
          {% if students.has_prev %}
```

```
                <li class="page-item">
                    <a class="page-link" href="{{ url_for('students', page=students.prev_num) }}">Previous</a>
                </li>
            {% endif %}
            {% for page_num in range(1, students.pages + 1) %}
                <li class="page-item {% if page_num == students.page %}active{% endif %}">
                    <a class="page-link" href="{{ url_for('students', page=page_num) }}">{{ page_num }}</a>
                </li>
            {% endfor %}
            {% if students.has_next %}
                <li class="page-item">
                    <a class="page-link" href="{{ url_for('students', page=students.next_num) }}">Next</a>
                </li>
            {% endif %}
        </ul>
    </nav>
</div>
</div>
</div>

<!-- Delete Confirmation Modals -->
{% for student in students.items %}
<div class="modal fade" id="deleteModal{{ student.id }}" tabindex="-1" aria-labelledby="deleteModalLabel{{ student.id }}" aria-hidden="true">
    <div class="modal-dialog">
```

```
      <div class="modal-content">
        <div class="modal-header">
          <h5 class="modal-title" id="deleteModalLabel{{ student.id }}">Confirm Delete</h5>
          <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
        </div>
        <div class="modal-body">
          Are you sure you want to delete student <strong>{{ student.name }}</strong>? This action cannot be undone.
        </div>
        <div class="modal-footer">
          <button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Cancel</button>
          <form action="{{ url_for('delete_student', student_id=student.id) }}" method="POST">
            {{ form.csrf_token }}
            <button type="submit" class="btn btn-danger">Delete</button>
          </form>
        </div>
      </div>
    </div>
  </div>
{% endfor %}

<!-- Bootstrap Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
{% endblock %}
```

however it is gone from verify_otp

---

look at the ss

there are double flash msgs showing

---

okay

now let's clean our project folder

there are many files and folders in it which are not in use now

should i send u the ss of my file tree?

---

wait

hostel.db was sqlite db

and we're using mysql rds now

should i still keep it?

---

how to check the subnet if we've the repo in ecr and the service is running in ecs

---

but the alb is connected to the public subnets and the ecs is in private subnet

what do we show in the public subnet in the architecture diagram?

---

is this correct?

just ignore the ecs present in public subnet

okay

since we set the alb on ip addresses mode instead of instance why?

what does this mean?

---

and should we add asg?

---

what did we use for our otp thing?

when i record a payemtn:

Internal Server Error

The server encountered an internal error and was unable to complete your request.
Either the server is overloaded or there is an error in the application.

but then refresh the page and the payment is recorded

---

give the final edited file

---

no let's print the error in flash for now

---

look at this function too

there's no variable named allowed_file

---

same error came when i recorded the payment:

An error occurred while recording the payment.

Error: property 'fee_status' of 'Student' object has no setter