Task 5

**1. Project Structure**

plaintext

CopyEdit

hotel-booking-api/

│

├── app/

│ ├── \_\_init\_\_.py

│ ├── models.py

│ ├── routes.py

│ ├── utils.py

│ └── config.py

│

├── migrations/

│

├── requirements.txt

│

└── run.py

**2. requirements.txt**

plaintext

CopyEdit

Flask

Flask-SQLAlchemy

Flask-Migrate

Flask-JWT-Extended

psycopg2

**3. Initialize the Flask App (app/\_\_init\_\_.py)**

python

CopyEdit

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

from flask\_jwt\_extended import JWTManager

db = SQLAlchemy()

migrate = Migrate()

jwt = JWTManager()

def create\_app():

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

# Configuration

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'postgresql://username:password@localhost/hotel\_db'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

app.config['JWT\_SECRET\_KEY'] = 'your\_secret\_key\_here' # Change to a secure key

# Initialize extensions

db.init\_app(app)

migrate.init\_app(app, db)

jwt.init\_app(app)

from .routes import bp as routes\_bp

app.register\_blueprint(routes\_bp)

return app

**4. Define Models (app/models.py)**

python

CopyEdit

from datetime import datetime

from app import db

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

password = db.Column(db.String(120), nullable=False)

rooms = db.relationship('Room', backref='owner', lazy=True)

bookings = db.relationship('Booking', backref='user', lazy=True)

class Room(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(120), nullable=False)

description = db.Column(db.Text, nullable=False)

price\_per\_night = db.Column(db.Float, nullable=False)

available\_from = db.Column(db.DateTime, nullable=False)

available\_to = db.Column(db.DateTime, nullable=False)

owner\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

class Booking(db.Model):

id = db.Column(db.Integer, primary\_key=True)

check\_in = db.Column(db.DateTime, nullable=False)

check\_out = db.Column(db.DateTime, nullable=False)

room\_id = db.Column(db.Integer, db.ForeignKey('room.id'), nullable=False)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

room = db.relationship('Room', backref='bookings', lazy=True)

**5. Define Routes (app/routes.py)**

python

CopyEdit

from flask import Blueprint, request, jsonify

from app import db

from app.models import User, Room, Booking

from flask\_jwt\_extended import create\_access\_token, jwt\_required, get\_jwt\_identity

from datetime import datetime

from sqlalchemy import or\_

bp = Blueprint('routes', \_\_name\_\_)

# Register a new user

@bp.route('/register', methods=['POST'])

def register\_user():

data = request.get\_json()

username = data['username']

password = data['password'] # Ideally hashed

user = User(username=username, password=password)

db.session.add(user)

db.session.commit()

return jsonify(message="User registered successfully"), 201

# Login and generate JWT token

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

def login\_user():

data = request.get\_json()

username = data['username']

password = data['password']

user = User.query.filter\_by(username=username, password=password).first() # Password check should be hashed

if user:

access\_token = create\_access\_token(identity=user.id)

return jsonify(access\_token=access\_token)

return jsonify(message="Invalid credentials"), 401

# Create a new room listing

@bp.route('/rooms', methods=['POST'])

@jwt\_required()

def create\_room():

data = request.get\_json()

current\_user = get\_jwt\_identity()

room = Room(

name=data['name'],

description=data['description'],

price\_per\_night=data['price\_per\_night'],

available\_from=datetime.strptime(data['available\_from'], '%Y-%m-%d'),

available\_to=datetime.strptime(data['available\_to'], '%Y-%m-%d'),

owner\_id=current\_user

)

db.session.add(room)

db.session.commit()

return jsonify(message="Room created successfully"), 201

# Edit a room listing

@bp.route('/rooms/<int:room\_id>', methods=['PUT'])

@jwt\_required()

def edit\_room(room\_id):

data = request.get\_json()

current\_user = get\_jwt\_identity()

room = Room.query.filter\_by(id=room\_id, owner\_id=current\_user).first()

if not room:

return jsonify(message="Room not found or not authorized"), 404

room.name = data['name']

room.description = data['description']

room.price\_per\_night = data['price\_per\_night']

room.available\_from = datetime.strptime(data['available\_from'], '%Y-%m-%d')

room.available\_to = datetime.strptime(data['available\_to'], '%Y-%m-%d')

db.session.commit()

return jsonify(message="Room updated successfully")

# Delete a room listing

@bp.route('/rooms/<int:room\_id>', methods=['DELETE'])

@jwt\_required()

def delete\_room(room\_id):

current\_user = get\_jwt\_identity()

room = Room.query.filter\_by(id=room\_id, owner\_id=current\_user).first()

if not room:

return jsonify(message="Room not found or not authorized"), 404

db.session.delete(room)

db.session.commit()

return jsonify(message="Room deleted successfully")

# Search available rooms

@bp.route('/search\_rooms', methods=['GET'])

def search\_rooms():

check\_in = request.args.get('check\_in')

check\_out = request.args.get('check\_out')

check\_in\_date = datetime.strptime(check\_in, '%Y-%m-%d')

check\_out\_date = datetime.strptime(check\_out, '%Y-%m-%d')

rooms = Room.query.filter(

Room.available\_from <= check\_in\_date,

Room.available\_to >= check\_out\_date

).all()

results = []

for room in rooms:

results.append({

'name': room.name,

'description': room.description,

'price\_per\_night': room.price\_per\_night,

'available\_from': room.available\_from,

'available\_to': room.available\_to

})

return jsonify(rooms=results)

# Booking a room

@bp.route('/book\_room/<int:room\_id>', methods=['POST'])

@jwt\_required()

def book\_room(room\_id):

data = request.get\_json()

check\_in = datetime.strptime(data['check\_in'], '%Y-%m-%d')

check\_out = datetime.strptime(data['check\_out'], '%Y-%m-%d')

current\_user = get\_jwt\_identity()

room = Room.query.get(room\_id)

if not room:

return jsonify(message="Room not found"), 404

if room.available\_from <= check\_in and room.available\_to >= check\_out:

booking = Booking(

check\_in=check\_in,

check\_out=check\_out,

room\_id=room.id,

user\_id=current\_user

)

db.session.add(booking)

db.session.commit()

return jsonify(message="Room booked successfully"), 201

return jsonify(message="Room is not available for the selected dates"), 400

**6. Input Validation and Error Handling**

In production, you should also ensure input validation using libraries like Marshmallow for serializing data, and ensure that exceptions are handled gracefully, particularly for database queries.

**7. Running the App (run.py)**

python

CopyEdit

from app import create\_app

app = create\_app()

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

**8. Database Migrations**

Make sure to run migrations for creating the necessary tables:

bash

CopyEdit

flask db init

flask db migrate

flask db upgrade