# app.py

from flask import Flask, request, jsonify, session, send\_from\_directory

from werkzeug.security import check\_password\_hash, generate\_password\_hash

from werkzeug.utils import secure\_filename # NEW: Import for secure file uploads

from pymongo import MongoClient

from bcrypt import hashpw, checkpw, gensalt

from bson.binary import Binary

from flask\_cors import CORS

from datetime import datetime

from bson.objectid import ObjectId

import os # NEW: Import for file system operations

from functools import wraps # NEW: Import for decorators

import uuid

# --- Configuration ---

MONGO\_URI = "mongodb+srv://sakshi:gaurinde@cluster0.vpbqv.mongodb.net/sp\_db?retryWrites=true&w=majority&appName=Cluster0"

DB\_NAME = "sp\_db"

STUDENT\_COLLECTION = "students"

ADMIN\_COLLECTION = "admins"

COURSE\_COLLECTION = "courses"

STUDY\_MATERIALS\_COLLECTION = "study\_materials"

USER\_NOTES\_COLLECTION = "user\_notes"

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

app.config['SECRET\_KEY'] = 'a\_very\_long\_and\_random\_secret\_key\_here\_that\_is\_unique\_and\_not\_guessable'

CORS(app, resources={r"/api/\*": {"origins": "http://localhost:5173"}}, supports\_credentials=True)

# --- File Upload Configuration for Admin Materials ---

# Define the upload folder. It's good practice to place it within a static directory.

# Ensure this directory exists! Example: your\_project\_root/static/uploads/admin\_materials

UPLOAD\_MATERIALS\_FOLDER = os.path.join(app.root\_path, 'static', 'uploads', 'admin\_materials')

# Create the upload folder if it doesn't exist

if not os.path.exists(UPLOAD\_MATERIALS\_FOLDER):

    os.makedirs(UPLOAD\_MATERIALS\_FOLDER)

app.config['UPLOAD\_MATERIALS\_FOLDER'] = UPLOAD\_MATERIALS\_FOLDER

# Allowed extensions for admin-uploaded materials

ALLOWED\_EXTENSIONS = {'pdf', 'png', 'jpg', 'jpeg', 'gif', 'doc', 'docx', 'txt'} # Added document extensions

# Helper function to check allowed file extensions

def allowed\_file(filename):

    return '.' in filename and \

           filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

client = MongoClient(MONGO\_URI)

db = client.get\_database(DB\_NAME)

students\_collection = db[STUDENT\_COLLECTION]

admins\_collection = db[ADMIN\_COLLECTION]

courses\_collection = db[COURSE\_COLLECTION]

study\_materials\_collection = db[STUDY\_MATERIALS\_COLLECTION]

user\_notes\_collection = db[USER\_NOTES\_COLLECTION] # Not used in this app.py, but kept for consistency

@app.route('/')

def hello\_world():

    return 'Hello, World!'

@app.route('/test\_db')

def test\_database\_connection():

    try:

        client.admin.command('ping')

        return "Successfully connected to MongoDB Atlas!"

    except Exception as e:

        return f"Could not connect to MongoDB Atlas: {e}"

@app.route('/api/auth/signup/student', methods=['POST'])

def signup\_student():

    data = request.get\_json()

    full\_name = data.get('fullName')

    email = data.get('email')

    password = data.get('password')

    if not full\_name or not email or not password:

        return jsonify({'message': 'Missing required fields'}), 400

    if students\_collection.find\_one({'email': email}):

        return jsonify({'message': 'Email already exists'}), 409

    salt = gensalt()

    hashed\_password = hashpw(password.encode('utf-8'), salt)

    student\_data = {'fullName': full\_name, 'email': email, 'password': hashed\_password, 'salt': salt}

    students\_collection.insert\_one(student\_data)

    return jsonify({'message': 'Student registered successfully'}), 201

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

def login\_student():

    data = request.get\_json()

    email = data.get('email')

    password = data.get('password')

    if not email or not password:

        return jsonify({'message': 'Missing email or password'}), 400

    student = students\_collection.find\_one({'email': email})

    if student:

        stored\_password = student.get('password')

        if isinstance(stored\_password, bytes) and checkpw(password.encode('utf-8'), stored\_password):

            session['user\_id'] = str(student['\_id'])

            session['username'] = student['fullName']

            session['role'] = 'student'

            return jsonify({'message': 'Student login successful', 'role': 'student'}), 200

    return jsonify({'message': 'Invalid credentials'}), 401

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

def login\_admin():

    data = request.get\_json()

    username = data.get('username')

    password = data.get('password')

    if not username or not password:

        return jsonify({'message': 'Missing username or password'}), 400

    admin = admins\_collection.find\_one({'username': username})

    if admin:

        stored\_password\_hash = admin.get('password')

        if stored\_password\_hash and isinstance(stored\_password\_hash, bytes) and checkpw(password.encode('utf-8'), stored\_password\_hash):

            session['user\_id'] = str(admin['\_id'])

            session['username'] = admin['username']

            session['role'] = 'admin'

            return jsonify({'message': 'Admin login successful', 'role': 'admin'}), 200

    return jsonify({'message': 'Invalid credentials'}), 401

@app.route('/api/check\_auth', methods=['GET'])

def check\_auth():

    if 'user\_id' in session:

        return jsonify({'isAuthenticated': True, 'role': session['role'], 'username': session['username']}), 200

    return jsonify({'isAuthenticated': False}), 200

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

def logout\_user():

    session.clear()

    return jsonify({"message": "Successfully logged out"}), 200

@app.route('/api/admin/setup', methods=['POST'])

def admin\_setup():

    if admins\_collection.find\_one({'username': 'superadmin'}):

        return jsonify({'message': 'Admin user already exists'}), 409

    hashed\_password = hashpw('supersecurepassword'.encode('utf-8'), gensalt())

    admins\_collection.insert\_one({'username': 'superadmin', 'password': hashed\_password, 'role': 'admin'})

    return jsonify({'message': 'Superadmin created successfully'}), 201

@app.route('/api/courses', methods=['GET'])

def get\_courses():

    courses = list(courses\_collection.find({}, {'\_id': 0, 'code': 1, 'title': 1}))

    return jsonify(courses), 200

@app.route('/api/courses/<course\_code>/years', methods=['GET'])

def get\_years(course\_code):

    course = courses\_collection.find\_one({'code': course\_code}, {'\_id': 0, 'years': 1})

    if course:

        years = [y['year'] for y in course.get('years', [])]

        return jsonify(years), 200

    return jsonify([]), 404

@app.route('/api/courses/<course\_code>/years/<int:year>/semesters', methods=['GET'])

def get\_semesters(course\_code, year):

    course = courses\_collection.find\_one({'code': course\_code}, {'\_id': 0, 'years': 1})

    if course:

        year\_data = next((y for y in course.get('years', []) if y['year'] == year), None)

        if year\_data:

            semesters = [s['semester'] for s in year\_data.get('semesters', [])]

            return jsonify(semesters), 200

    return jsonify([]), 404

@app.route('/api/courses/<course\_code>/years/<int:year>/semesters/<int:semester>/subjects', methods=['GET'])

def get\_subjects\_route(course\_code, year, semester):

    course = courses\_collection.find\_one({'code': course\_code}, {'\_id': 0, 'years': 1})

    if course:

        year\_data = next((y for y in course.get('years', []) if y['year'] == year), None)

        if year\_data:

            semester\_data = next((s for s in year\_data.get('semesters', []) if s['semester'] == semester), None)

            if semester\_data:

                return jsonify(semester\_data.get('subjects', [])), 200

    return jsonify([]), 404

@app.route('/api/search/subjects', methods=['GET'])

def search\_subjects():

    search\_term = request.args.get('q', '').strip()

    if not search\_term:

        return jsonify([]), 200

    pipeline = [

        {"$unwind": "$years"},

        {"$unwind": "$years.semesters"},

        {"$unwind": "$years.semesters.subjects"},

        {"$match": {"years.semesters.subjects": {"$regex": search\_term, "$options": "i"}}},

        {"$project": {

            "\_id": 0,

            "subjectName": "$years.semesters.subjects",

            "courseName": "$title",

            "courseCode": "$code",

            "year": "$years.year",

            "semester": "$years.semesters.semester"

        }}

    ]

    results = list(courses\_collection.aggregate(pipeline))

    return jsonify(results), 200

def is\_admin():

    return session.get('role') == 'admin'

def admin\_required(f):

    @wraps(f) # Use @wraps to preserve function metadata

    def wrapper(\*args, \*\*kwargs):

        if not is\_admin():

            return jsonify({'message': 'Forbidden: Admin access required'}), 403

        return f(\*args, \*\*kwargs)

    return wrapper

# --- Admin Material Management ---

@app.route('/api/admin/materials/add', methods=['POST'])

@admin\_required

def admin\_add\_material():

    data = request.get\_json()

    title = data.get('title')

    course\_code = data.get('courseCode')

    year = data.get('year')

    semester = data.get('semester')

    subject = data.get('subject')

    material\_format = data.get('materialFormat', '')

    material\_category = data.get('materialCategory', '')

    content\_url = data.get('contentUrl')

    if not all([title, course\_code, year, semester, subject, material\_format, material\_category]):

        return jsonify({'message': 'Missing required fields'}), 400

    # Ensure material\_category is valid

    if material\_category not in ['syllabus', 'notes', 'paper']:

        return jsonify({'message': 'Invalid material Category. Must be one of: syllabus, notes, paper'}), 400

    # This endpoint now handles only URL-based formats (Video, Link)

    if material\_format not in ['Video', 'Link']:

        return jsonify({'message': 'Invalid material format for this endpoint. Use /api/admin/materials/upload for files.'}), 400

    if not content\_url:

        return jsonify({'message': f"Content URL is required for {material\_format} format"}), 400

    # Basic validation for year and semester if they are expected as integers

    try:

        year = int(year)

        semester = int(semester)

    except ValueError:

        return jsonify({'message': 'Year and Semester must be valid numbers.'}), 400

    material\_doc = {

        'title': title,

        'courseCode': course\_code,

        'year': year,

        'semester': semester,

        'subject': subject,

        'materialFormat': material\_format,

        'materialCategory': material\_category,

        'contentUrl': content\_url,

        'uploadedBy': session.get('username'),

        'uploadedAt': datetime.utcnow()

    }

    try:

        study\_materials\_collection.insert\_one(material\_doc)

        return jsonify({'message': 'Material added successfully'}), 201

    except Exception as e:

        print(f"Error adding material: {e}")

        return jsonify({'message': 'Failed to add material', 'error': str(e)}), 500

# --- REINSTATED: Admin Material Upload Endpoint ---

@app.route('/api/admin/materials/upload', methods=['POST'])

@admin\_required

def upload\_admin\_material():

    # Check if a file was sent

    if 'file' not in request.files:

        return jsonify({'message': 'No file part in the request'}), 400

    file = request.files['file']

    # If the user does not select a file, the browser submits an empty file without a filename.

    if file.filename == '':

        return jsonify({'message': 'No selected file'}), 400

    # Ensure file is allowed and process it

    if file and allowed\_file(file.filename):

        try:

            # Get other form data

            title = request.form.get('title')

            course\_code = request.form.get('courseCode')

            year = request.form.get('year')

            semester = request.form.get('semester')

            subject = request.form.get('subject')

            material\_format = request.form.get('materialFormat')

            material\_category = request.form.get('materialCategory')

            if not all([title, course\_code, year, semester, subject, material\_format, material\_category]):

                return jsonify({'message': 'Missing required form data for material'}), 400

            if material\_category not in ['syllabus', 'notes', 'paper']: # Validate category for uploads too

                return jsonify({'message': 'Invalid material Category. Must be one of: syllabus, notes, paper'}), 400

            # Secure the filename before saving

            filename = f"{uuid.uuid4().hex}\_{secure\_filename(file.filename)}"

            file\_path = os.path.join(app.config['UPLOAD\_MATERIALS\_FOLDER'], filename)

            file.save(file\_path)

            # Construct the URL to access the uploaded file

            content\_url = f"/static/uploads/admin\_materials/{filename}"

            # Save material data to MongoDB

            material\_entry = {

                'title': title,

                'courseCode': course\_code,

                'year': int(year), # Ensure year is stored as int

                'semester': int(semester), # Ensure semester is stored as int

                'subject': subject,

                'materialFormat': material\_format,

                'materialCategory': material\_category,

                'contentUrl': content\_url, # Store the URL to the uploaded file

                'fileName': filename, # Store original filename for potential future use (e.g., deletion)

                'uploadedBy': session.get('username'), # Assuming admin username is in session

                'uploadedAt': datetime.utcnow()

            }

            inserted\_result = study\_materials\_collection.insert\_one(material\_entry)

            material\_entry['\_id'] = str(inserted\_result.inserted\_id) # Convert ObjectId to string for JSON serialization

            return jsonify({'message': 'Material uploaded and added successfully', 'material': material\_entry}), 201

        except ValueError as ve: # Catch specific ValueError for int conversion

            print(f"Validation error: {ve}")

            return jsonify({'message': f'Invalid data provided: {ve}'}), 400

        except Exception as e:

            print(f"Error uploading material: {e}")

            return jsonify({'message': 'Server error during file upload', 'error': str(e)}), 500

    else:

        return jsonify({'message': 'File type not allowed or no file provided'}), 400

# --- Get Materials for Student View ---

@app.route('/api/materials/<string:course\_code>/<int:year>/<int:semester>/<string:subject>', methods=['GET'])

def get\_study\_materials(course\_code, year, semester, subject):

    try:

        material\_format = request.args.get('materialFormat')

        material\_category = request.args.get('materialCategory')

        query = {

            'courseCode': course\_code,

            'year': year,

            'semester': semester,

            'subject': subject

        }

        if material\_format:

            query['materialFormat'] = material\_format

        if material\_category:

            query['materialCategory'] = material\_category # match DB case

        materials\_cursor = study\_materials\_collection.find(query)

        materials\_list = []

        for material in materials\_cursor:

            material['\_id'] = str(material['\_id'])

            if isinstance(material.get('uploadedAt'), datetime):

                material['uploadedAt'] = material['uploadedAt'].isoformat()

            materials\_list.append(material)

        return jsonify(materials\_list), 200

    except Exception as e:

        print(f"Error retrieving materials: {e}")

        return jsonify({'message': 'Failed to retrieve materials', 'error': str(e)}), 500

# --- Get Materials for Admin View (with filters) ---

@app.route('/api/admin/materials', methods=['GET'])

@admin\_required

def admin\_get\_materials():

    course\_code = request.args.get('courseCode')

    year = request.args.get('year')

    semester = request.args.get('semester')

    subject = request.args.get('subject')

    query = {}

    if course\_code:

        query['courseCode'] = course\_code

    if year:

        try:

            query['year'] = int(year) # Ensure year is int

        except ValueError:

            return jsonify({'message': 'Invalid year format, must be integer'}), 400

    if semester:

        try:

            query['semester'] = int(semester) # Ensure semester is int

        except ValueError:

            return jsonify({'message': 'Invalid semester format, must be integer'}), 400

    if subject:

        query['subject'] = subject

    try:

        materials = list(study\_materials\_collection.find(query))

        for material in materials:

            material['\_id'] = str(material['\_id']) # Convert ObjectId to string for JSON serialization

            if isinstance(material.get('uploadedAt'), datetime): # Convert datetime to string for JSON

                material['uploadedAt'] = material['uploadedAt'].isoformat()

        return jsonify(materials), 200

    except Exception as e:

        print(f"Error fetching materials for admin: {e}")

        return jsonify({'message': 'Failed to fetch materials', 'error': str(e)}), 500

# --- Delete Material ---

@app.route('/api/admin/materials/<string:material\_id>', methods=['DELETE'])

@admin\_required

def admin\_delete\_material(material\_id):

    try:

        # Validate material\_id as a valid ObjectId

        if not ObjectId.is\_valid(material\_id):

            return jsonify({'message': 'Invalid Material ID format'}), 400

        # Before deleting from DB, check if it's a file and delete the file from server

        material\_doc = study\_materials\_collection.find\_one({'\_id': ObjectId(material\_id)})

        if material\_doc and 'fileName' in material\_doc and material\_doc.get('materialFormat') in ['PDF', 'Image', 'Document']:

            file\_to\_delete\_path = os.path.join(app.config['UPLOAD\_MATERIALS\_FOLDER'], material\_doc['fileName'])

            if os.path.exists(file\_to\_delete\_path):

                try:

                    os.remove(file\_to\_delete\_path)

                    print(f"Deleted file from server: {file\_to\_delete\_path}")

                except Exception as e:

                    print(f"Error deleting file from server: {e}")

                    # Log error but proceed with DB deletion

        result = study\_materials\_collection.delete\_one({'\_id': ObjectId(material\_id)})

        if result.deleted\_count == 1:

            return jsonify({'message': 'Material deleted successfully'}), 200

        else:

            return jsonify({'message': 'Material not found'}), 404

    except Exception as e:

        print(f"Error deleting material: {e}")

        return jsonify({'message': 'Failed to delete material', 'error': str(e)}), 500

# --- Update Material ---

@app.route('/api/admin/materials/<string:material\_id>', methods=['PUT'])

@admin\_required

def admin\_update\_material(material\_id):

    data = request.get\_json()

    # Validate material\_id as a valid ObjectId

    if not ObjectId.is\_valid(material\_id):

        return jsonify({'message': 'Invalid Material ID format'}), 400

    # Fields that can be updated

    update\_fields = [

        'title', 'materialFormat', 'materialCategory', 'contentUrl',

        'courseCode', 'year', 'semester', 'subject'

    ]

    update\_doc = {}

    for field in update\_fields:

        if field in data: # Only update if the field is present in the request body

            # Specific type conversion for year/semester if they are updated

            if field in ['year', 'semester']:

                try:

                    update\_doc[field] = int(data[field])

                except ValueError:

                    return jsonify({'message': f"Invalid format for {field}. Must be an integer."}), 400

            else:

                update\_doc[field] = data[field]

    if not update\_doc:

        return jsonify({'message': 'No fields provided for update'}), 400

    # Basic validation for materialFormat and associated content

    if 'materialFormat' in update\_doc:

        material\_format = update\_doc['materialFormat']

        # If materialFormat is changing to a URL-based type, contentUrl becomes required

        if material\_format in ['Video', 'Link'] and ('contentUrl' not in data or not data['contentUrl']):

            return jsonify({'message': f"Content URL is required for {material\_format} format"}), 400

        # If materialFormat is changing to a file-based type, ensure 'fileName' is handled in the future if updates are made through PUT

        # For now, PUT is expected to update metadata, not re-upload files.

        if material\_format in ['PDF', 'Image', 'Document'] and 'contentUrl' not in data:

            # If a file-based material is being updated, it should retain its contentUrl.

            # If the user wants to \*change\* the file, they'd need a separate upload mechanism.

            # Here, we assume contentUrl won't be empty for these types if it's not being changed

            pass # No strict validation for contentUrl on file types if it's not being changed

    # Also validate new category if it's provided

    if 'materialCategory' in update\_doc:

        if update\_doc['materialCategory'] not in ['syllabus', 'notes', 'paper']:

            return jsonify({'message': 'Invalid material Category. Must be one of: syllabus, notes, paper'}), 400

    try:

        result = study\_materials\_collection.update\_one(

            {'\_id': ObjectId(material\_id)},

            {'$set': update\_doc}

        )

        if result.matched\_count == 0:

            return jsonify({'message': 'Material not found'}), 404

        elif result.modified\_count == 0:

            return jsonify({'message': 'No changes made to material (data was identical or material not found)'}), 200

        else:

            return jsonify({'message': 'Material updated successfully'}), 200

    except Exception as e:

        print(f"Error updating material: {e}")

        return jsonify({'message': 'Failed to update material', 'error': str(e)}), 500

# --- Static File Serving ---

@app.route('/static/uploads/admin\_materials/<filename>')

def serve\_admin\_material\_file(filename):

    # This route serves files from the UPLOAD\_MATERIALS\_FOLDER

    # Ensure this folder is correctly configured and accessible

    print(f"Attempting to serve file: {filename} from {app.config['UPLOAD\_MATERIALS\_FOLDER']}")

    try:

        return send\_from\_directory(app.config['UPLOAD\_MATERIALS\_FOLDER'], filename)

    except FileNotFoundError:

        print(f"File not found: {filename} in {app.config['UPLOAD\_MATERIALS\_FOLDER']}")

        # You might want to log this or return a more specific error,

        # or handle it with an @app.errorhandler(404) if it should redirect.

        return jsonify({'message': 'File not found'}), 404

#PASSWORD CHANGE

@app.route('/api/admin/change-password', methods=['POST'])

@admin\_required # Your admin login decorator

def change\_password():

    data = request.get\_json()

    current\_password = data.get('currentPassword')

    new\_password = data.get('newPassword')

    if not current\_password or not new\_password:

        return jsonify({'message': 'Missing password fields'}), 400

    user\_id = session.get('user\_id')

    if not user\_id:

        return jsonify({'message': 'Unauthorized'}), 401

    try:

        # Assuming user IDs in session are strings, convert to ObjectId for MongoDB lookup

        user = admins\_collection.find\_one({'\_id': ObjectId(user\_id)})

    except Exception as e:

        # Log the error for debugging purposes

        print(f"Error fetching user in change\_password: {e}")

        return jsonify({'message': 'Internal server error while fetching user data.'}), 500

    if not user:

        return jsonify({'message': 'User not found'}), 404

    stored\_password\_hash = user.get('password') # Use .get() for safety

    if stored\_password\_hash is None:

        return jsonify({'message': 'Password hash not found for user.'}), 500

    if isinstance(stored\_password\_hash, str):

        stored\_password\_hash\_bytes = stored\_password\_hash.encode('utf-8')

    elif isinstance(stored\_password\_hash, bytes):

        stored\_password\_hash\_bytes = stored\_password\_hash

    else:

        print(f"Unexpected type for stored password hash: {type(stored\_password\_hash)}")

        return jsonify({'message': 'Invalid stored password hash format.'}), 500

    # Verify current password using bcrypt.checkpw

    if not checkpw(current\_password.encode('utf-8'), stored\_password\_hash\_bytes):

        return jsonify({'message': 'Incorrect current password'}), 403

    # Basic new password validation (you might have more robust frontend validation)

    if len(new\_password) < 6: # Example: Minimum password length

        return jsonify({'message': 'New password must be at least 6 characters long.'}), 400

    # Hash the new password using bcrypt.hashpw

    hashed\_new\_password = Binary(hashpw(new\_password.encode('utf-8'), gensalt()))

    try:

        admins\_collection.update\_one(

            {'\_id': ObjectId(user\_id)},

            {'$set': {'password': hashed\_new\_password}}

        )

        return jsonify({'message': 'Password changed successfully'}), 200

    except Exception as e:

        print(f"Error updating password in change\_password: {e}")

        return jsonify({'message': 'Internal server error while updating password.'}), 500

#USERNAME CHANGE

@app.route('/api/admin/change-username', methods=['POST'])

@admin\_required # Your admin login decorator

def change\_username():

    data = request.get\_json()

    new\_username = data.get('newUsername')

    if not new\_username:

        return jsonify({'message': 'Missing new username'}), 400

    user\_id = session.get('user\_id')

    if not user\_id:

        return jsonify({'message': 'Unauthorized'}), 401

    user = admins\_collection.find\_one({'\_id': ObjectId(user\_id)})

    if not user:

        return jsonify({'message': 'User not found'}), 404

    # Check if username already exists (optional but recommended)

    if admins\_collection.find\_one({'username': new\_username, '\_id': {'$ne': ObjectId(user\_id)}}):

        return jsonify({'message': 'Username already taken'}), 409

    # Update username

    admins\_collection.update\_one(

        {'\_id': ObjectId(user\_id)},

        {'$set': {'username': new\_username}}

    )

    return jsonify({'message': 'Username updated successfully'}), 200

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

    app.run(debug=True, host='0.0.0.0') # Make it accessible from your frontend