**mplémentation d'une API de gestion des notes**

Je vais créer les modèles, contrôleurs et routes nécessaires pour votre système de gestion des notes avec sessions normales et de rattrapage.

**1. Modèles**

Commençons par compléter le modèle Note et créer les modèles supplémentaires nécessaires.

from app import db

from datetime import datetime

from app.models.user import User, UserRole

class SessionType(enum.Enum):

NORMALE = 'normale'

RATTRAPAGE = 'rattrapage'

class Semestre(enum.Enum):

S1 = 'Semestre 1'

S2 = 'Semestre 2'

S3 = 'Semestre 3'

S4 = 'Semestre 4'

S5 = 'Semestre 5'

S6 = 'Semestre 6'

class Matiere(db.Model):

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

nom = db.Column(db.String(100), nullable=False)

code = db.Column(db.String(20), unique=True, nullable=False)

coefficient = db.Column(db.Float, default=1.0)

semestre\_id = db.Column(db.Enum(Semestre), nullable=False)

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

# Relations

enseignant = db.relationship('User', backref='matieres\_enseignees')

notes = db.relationship('Note', backref='matiere', lazy=True, cascade="all, delete-orphan")

def \_\_repr\_\_(self):

return f'<Matiere {self.code}: {self.nom}>'

class Note(db.Model):

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

valeur = db.Column(db.Float, nullable=False)

date\_ajout = db.Column(db.DateTime, default=datetime.utcnow)

commentaire = db.Column(db.Text, nullable=True)

session\_type = db.Column(db.Enum(SessionType), default=SessionType.NORMALE)

# Clés étrangères

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

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

# Relations

etudiant = db.relationship('User', backref='notes')

def \_\_repr\_\_(self):

return f'<Note: {self.valeur} pour {self.etudiant.username} en {self.matiere.nom}>'

class Message(db.Model):

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

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

date\_envoi = db.Column(db.DateTime, default=datetime.utcnow)

lu = db.Column(db.Boolean, default=False)

# Clés étrangères

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

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

# Relations

expediteur = db.relationship('User', foreign\_keys=[expediteur\_id], backref='messages\_envoyes')

destinataire = db.relationship('User', foreign\_keys=[destinataire\_id], backref='messages\_recus')

def \_\_repr\_\_(self):

return f'<Message de {self.expediteur.username} à {self.destinataire.username}>'

Copy

Apply

app\models\notes.py

**2. Contrôleurs**

Maintenant, créons les contrôleurs pour gérer les notes, matières et messages.

from flask import jsonify, request

from app import db

from app.models.notes import Matiere, Semestre

from app.models.user import User, UserRole

def get\_all\_matieres():

matieres = Matiere.query.all()

result = []

for matiere in matieres:

result.append({

'id': matiere.id,

'nom': matiere.nom,

'code': matiere.code,

'coefficient': matiere.coefficient,

'semestre': matiere.semestre\_id.value,

'enseignant': matiere.enseignant.username if matiere.enseignant else None

})

return jsonify(result), 200

def get\_matiere(matiere\_id):

matiere = Matiere.query.get\_or\_404(matiere\_id)

return jsonify({

'id': matiere.id,

'nom': matiere.nom,

'code': matiere.code,

'coefficient': matiere.coefficient,

'semestre': matiere.semestre\_id.value,

'enseignant': matiere.enseignant.username if matiere.enseignant else None

}), 200

def create\_matiere(data):

# Vérifier si le semestre est valide

if data['semestre'] not in Semestre.\_\_members\_\_:

return jsonify({'error': 'Semestre invalide'}), 400

# Vérifier si l'enseignant existe et est bien un enseignant

enseignant = None

if 'enseignant\_id' in data:

enseignant = User.query.get(data['enseignant\_id'])

if not enseignant or enseignant.role != UserRole.ENSEIGNANT:

return jsonify({'error': 'Enseignant invalide'}), 400

# Vérifier si le code matière existe déjà

if Matiere.query.filter\_by(code=data['code']).first():

return jsonify({'error': 'Code matière déjà utilisé'}), 400

new\_matiere = Matiere(

nom=data['nom'],

code=data['code'],

coefficient=data.get('coefficient', 1.0),

semestre\_id=Semestre[data['semestre']],

enseignant\_id=data.get('enseignant\_id')

)

db.session.add(new\_matiere)

db.session.commit()

return jsonify({'message': 'Matière créée avec succès', 'id': new\_matiere.id}), 201

def update\_matiere(matiere\_id, data):

matiere = Matiere.query.get\_or\_404(matiere\_id)

if 'nom' in data:

matiere.nom = data['nom']

if 'code' in data:

# Vérifier si le nouveau code n'est pas déjà utilisé par une autre matière

existing = Matiere.query.filter\_by(code=data['code']).first()

if existing and existing.id != matiere\_id:

return jsonify({'error': 'Code matière déjà utilisé'}), 400

matiere.code = data['code']

if 'coefficient' in data:

matiere.coefficient = data['coefficient']

if 'semestre' in data:

if data['semestre'] not in Semestre.\_\_members\_\_:

return jsonify({'error': 'Semestre invalide'}), 400

matiere.semestre\_id = Semestre[data['semestre']]

if 'enseignant\_id' in data:

enseignant = User.query.get(data['enseignant\_id'])

if not enseignant or enseignant.role != UserRole.ENSEIGNANT:

return jsonify({'error': 'Enseignant invalide'}), 400

matiere.enseignant\_id = data['enseignant\_id']

db.session.commit()

return jsonify({'message': 'Matière mise à jour avec succès'}), 200

def delete\_matiere(matiere\_id):

matiere = Matiere.query.get\_or\_404(matiere\_id)

db.session.delete(matiere)

db.session.commit()

return jsonify({'message': 'Matière supprimée avec succès'}), 200

Copy

Apply

app\controllers\matiere\_controller.py

from flask import jsonify, request

from app import db

from app.models.notes import Note, Matiere, SessionType

from app.models.user import User, UserRole

from sqlalchemy import and\_

def get\_notes\_by\_etudiant(etudiant\_id):

# Vérifier si l'étudiant existe

etudiant = User.query.get\_or\_404(etudiant\_id)

if etudiant.role != UserRole.ETUDIANT:

return jsonify({'error': 'Utilisateur non étudiant'}), 400

notes = Note.query.filter\_by(etudiant\_id=etudiant\_id).all()

result = []

for note in notes:

result.append({

'id': note.id,

'valeur': note.valeur,

'matiere': {

'id': note.matiere.id,

'nom': note.matiere.nom,

'code': note.matiere.code,

'coefficient': note.matiere.coefficient,

'semestre': note.matiere.semestre\_id.value

},

'session': note.session\_type.value,

'date\_ajout': note.date\_ajout.isoformat(),

'commentaire': note.commentaire

})

return jsonify(result), 200

def get\_notes\_by\_matiere(matiere\_id):

# Vérifier si la matière existe

matiere = Matiere.query.get\_or\_404(matiere\_id)

notes = Note.query.filter\_by(matiere\_id=matiere\_id).all()

result = []

for note in notes:

result.append({

'id': note.id,

'valeur': note.valeur,

'etudiant': {

'id': note.etudiant.id,

'username': note.etudiant.username

},

'session': note.session\_type.value,

'date\_ajout': note.date\_ajout.isoformat(),

'commentaire': note.commentaire

})

return jsonify(result), 200

def create\_note(data):

# Vérifier si l'étudiant existe et est bien un étudiant

etudiant = User.query.get(data['etudiant\_id'])

if not etudiant or etudiant.role != UserRole.ETUDIANT:

return jsonify({'error': 'Étudiant invalide'}), 400

# Vérifier si la matière existe

matiere = Matiere.query.get(data['matiere\_id'])

if not matiere:

return jsonify({'error': 'Matière invalide'}), 400

# Vérifier si la session est valide

if data['session'] not in SessionType.\_\_members\_\_:

return jsonify({'error': 'Type de session invalide'}), 400

# Vérifier si la note est dans la plage valide (0-20)

if not 0 <= data['valeur'] <= 20:

return jsonify({'error': 'La note doit être comprise entre 0 et 20'}), 400

# Vérifier si une note existe déjà pour cet étudiant dans cette matière et cette session

existing\_note = Note.query.filter(

and\_(

Note.etudiant\_id == data['etudiant\_id'],

Note.matiere\_id == data['matiere\_id'],

Note.session\_type == SessionType[data['session']]

)

).first()

if existing\_note:

return jsonify({'error': 'Une note existe déjà pour cet étudiant dans cette matière et cette session'}), 400

new\_note = Note(

valeur=data['valeur'],

commentaire=data.get('commentaire'),

session\_type=SessionType[data['session']],

etudiant\_id=data['etudiant\_id'],

matiere\_id=data['matiere\_id']

)

db.session.add(new\_note)

db.session.commit()

return jsonify({'message': 'Note ajoutée avec succès', 'id': new\_note.id}), 201

def update\_note(note\_id, data):

note = Note.query.get\_or\_404(note\_id)

if 'valeur' in data:

if not 0 <= data['valeur'] <= 20:

return jsonify({'error': 'La note doit être comprise entre 0 et 20'}), 400

note.valeur = data['valeur']

if 'commentaire' in data:

note.commentaire = data['commentaire']

if 'session' in data:

if data['session'] not in SessionType.\_\_members\_\_:

return jsonify({'error': 'Type de session invalide'}), 400

# Si on change de session, vérifier qu'il n'y a pas déjà une note pour cette session

if note.session\_type != SessionType[data['session']]:

existing\_note = Note.query.filter(

and\_(

Note.etudiant\_id == note.etudiant\_id,

Note.matiere\_id == note.matiere\_id,

Note.session\_type == SessionType[data['session']],

Note.id != note\_id

)

).first()

if existing\_note:

return jsonify({'error': 'Une note existe déjà pour cet étudiant dans cette matière et cette session'}), 400

note.session\_type = SessionType[data['session']]

db.session.commit()

return jsonify({'message': 'Note mise à jour avec succès'}), 200

def delete\_note(note\_id):

note = Note.query.get\_or\_404(note\_id)

db.session.delete(note)

db.session.commit()

return jsonify({'message': 'Note supprimée avec succès'}), 200

def get\_bulletin(etudiant\_id, semestre=None):

# Vérifier si l'étudiant existe

etudiant = User.query.get\_or\_404(etudiant\_id)

if etudiant.role != UserRole.ETUDIANT:

return jsonify({'error': 'Utilisateur non étudiant'}), 400

# Construire la requête de base

query = db.session.query(Note, Matiere).join(Matiere).filter(Note.etudiant\_id == etudiant\_id)

# Filtrer par semestre si spécifié

if semestre:

if semestre not in Semestre.\_\_members\_\_:

return jsonify({'error': 'Semestre invalide'}), 400

query = query.filter(Matiere.semestre\_id == Semestre[semestre])

# Récupérer les résultats

results = query.all()

# Organiser les résultats par semestre

bulletin = {}

total\_points = 0

Copy

Apply

app\controllers\note\_controller.py

continuer

PromptsClaude 3.7 Sonnet

**Context— none fetched**

Je vais continuer l'implémentation du système de gestion des notes.

def get\_bulletin(etudiant\_id, semestre=None):

# Vérifier si l'étudiant existe

etudiant = User.query.get\_or\_404(etudiant\_id)

if etudiant.role != UserRole.ETUDIANT:

return jsonify({'error': 'Utilisateur non étudiant'}), 400

# Construire la requête de base

query = db.session.query(Note, Matiere).join(Matiere).filter(Note.etudiant\_id == etudiant\_id)

# Filtrer par semestre si spécifié

if semestre:

if semestre not in Semestre.\_\_members\_\_:

return jsonify({'error': 'Semestre invalide'}), 400

query = query.filter(Matiere.semestre\_id == Semestre[semestre])

# Récupérer les résultats

results = query.all()

# Organiser les résultats par semestre

bulletin = {}

for note, matiere in results:

semestre\_key = matiere.semestre\_id.value

if semestre\_key not in bulletin:

bulletin[semestre\_key] = {

'matieres': [],

'moyenne': 0,

'total\_coefficients': 0,

'total\_points': 0

}

# Prendre la meilleure note entre session normale et rattrapage

existing\_matiere = next((m for m in bulletin[semestre\_key]['matieres']

if m['id'] == matiere.id), None)

if existing\_matiere:

# Si la matière existe déjà, garder la meilleure note

if note.valeur > existing\_matiere['note']:

existing\_matiere['note'] = note.valeur

existing\_matiere['points'] = note.valeur \* matiere.coefficient

existing\_matiere['session'] = note.session\_type.value

else:

# Ajouter la matière avec sa note

matiere\_info = {

'id': matiere.id,

'nom': matiere.nom,

'code': matiere.code,

'coefficient': matiere.coefficient,

'note': note.valeur,

'points': note.valeur \* matiere.coefficient,

'session': note.session\_type.value

}

bulletin[semestre\_key]['matieres'].append(matiere\_info)

# Calculer les moyennes pour chaque semestre

for semestre\_key, semestre\_data in bulletin.items():

total\_points = 0

total\_coefficients = 0

for matiere in semestre\_data['matieres']:

total\_points += matiere['points']

total\_coefficients += matiere['coefficient']

semestre\_data['total\_points'] = total\_points

semestre\_data['total\_coefficients'] = total\_coefficients

if total\_coefficients > 0:

semestre\_data['moyenne'] = round(total\_points / total\_coefficients, 2)

else:

semestre\_data['moyenne'] = 0

return jsonify(bulletin), 200

Copy

Apply

app\controllers\note\_controller.py

from flask import jsonify

from app import db

from app.models.notes import Message

from app.models.user import User

from datetime import datetime

def get\_messages\_received(user\_id):

user = User.query.get\_or\_404(user\_id)

messages = Message.query.filter\_by(destinataire\_id=user\_id).order\_by(Message.date\_envoi.desc()).all()

result = []

for message in messages:

result.append({

'id': message.id,

'contenu': message.contenu,

'date\_envoi': message.date\_envoi.isoformat(),

'lu': message.lu,

'expediteur': {

'id': message.expediteur.id,

'username': message.expediteur.username,

'role': message.expediteur.role.value

}

})

return jsonify(result), 200

def get\_messages\_sent(user\_id):

user = User.query.get\_or\_404(user\_id)

messages = Message.query.filter\_by(expediteur\_id=user\_id).order\_by(Message.date\_envoi.desc()).all()

result = []

for message in messages:

result.append({

'id': message.id,

'contenu': message.contenu,

'date\_envoi': message.date\_envoi.isoformat(),

'lu': message.lu,

'destinataire': {

'id': message.destinataire.id,

'username': message.destinataire.username,

'role': message.destinataire.role.value

}

})

return jsonify(result), 200

def send\_message(data):

# Vérifier si l'expéditeur existe

expediteur = User.query.get(data['expediteur\_id'])

if not expediteur:

return jsonify({'error': 'Expéditeur invalide'}), 400

# Vérifier si le destinataire existe

destinataire = User.query.get(data['destinataire\_id'])

if not destinataire:

return jsonify({'error': 'Destinataire invalide'}), 400

# Créer le message

new\_message = Message(

contenu=data['contenu'],

expediteur\_id=data['expediteur\_id'],

destinataire\_id=data['destinataire\_id'],

date\_envoi=datetime.utcnow(),

lu=False

)

db.session.add(new\_message)

db.session.commit()

return jsonify({'message': 'Message envoyé avec succès', 'id': new\_message.id}), 201

def mark\_as\_read(message\_id):

message = Message.query.get\_or\_404(message\_id)

message.lu = True

db.session.commit()

return jsonify({'message': 'Message marqué comme lu'}), 200

def delete\_message(message\_id):

message = Message.query.get\_or\_404(message\_id)

db.session.delete(message)

db.session.commit()

return jsonify({'message': 'Message supprimé avec succès'}), 200

Copy

Apply

app\controllers\message\_controller.py

**3. Routes**

Maintenant, créons les routes pour notre API.

from flask import Blueprint, request, jsonify

from app.controllers.note\_controller import (

get\_notes\_by\_etudiant, get\_notes\_by\_matiere, create\_note,

update\_note, delete\_note, get\_bulletin

)

from functools import wraps

note\_bp = Blueprint('note', \_\_name\_\_)

# Middleware pour vérifier le format JSON

def require\_json(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if not request.is\_json:

return jsonify({"error": "Le format JSON est requis"}), 400

return f(\*args, \*\*kwargs)

return decorated\_function

# Routes pour les notes

@note\_bp.route('/etudiants/<int:etudiant\_id>/notes', methods=['GET'])

def get\_etudiant\_notes(etudiant\_id):

return get\_notes\_by\_etudiant(etudiant\_id)

@note\_bp.route('/matieres/<int:matiere\_id>/notes', methods=['GET'])

def get\_matiere\_notes(matiere\_id):

return get\_notes\_by\_matiere(matiere\_id)

@note\_bp.route('/notes', methods=['POST'])

@require\_json

def add\_note():

return create\_note(request.json)

@note\_bp.route('/notes/<int:note\_id>', methods=['PUT'])

@require\_json

def modify\_note(note\_id):

return update\_note(note\_id, request.json)

@note\_bp.route('/notes/<int:note\_id>', methods=['DELETE'])

def remove\_note(note\_id):

return delete\_note(note\_id)

@note\_bp.route('/etudiants/<int:etudiant\_id>/bulletin', methods=['GET'])

def student\_bulletin(etudiant\_id):

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

return get\_bulletin(etudiant\_id, semestre)

Copy

Apply

app\routes\note\_routes.py

from flask import Blueprint, request, jsonify

from app.controllers.matiere\_controller import (

get\_all\_matieres, get\_matiere, create\_matiere,

update\_matiere, delete\_matiere

)

from functools import wraps

matiere\_bp = Blueprint('matiere', \_\_name\_\_)

# Middleware pour vérifier le format JSON

def require\_json(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if not request.is\_json:

return jsonify({"error": "Le format JSON est requis"}), 400

return f(\*args, \*\*kwargs)

return decorated\_function

# Routes pour les matières

@matiere\_bp.route('/matieres', methods=['GET'])

def list\_matieres():

return get\_all\_matieres()

@matiere\_bp.route('/matieres/<int:matiere\_id>', methods=['GET'])

def get\_single\_matiere(matiere\_id):

return get\_matiere(matiere\_id)

@matiere\_bp.route('/matieres', methods=['POST'])

@require\_json

def add\_matiere():

return create\_matiere(request.json)

@matiere\_bp.route('/matieres/<int:matiere\_id>', methods=['PUT'])

@require\_json

def modify\_matiere(matiere\_id):

return update\_matiere(matiere\_id, request.json)

@matiere\_bp.route('/matieres/<int:matiere\_id>', methods=['DELETE'])

def remove\_matiere(matiere\_id):

return delete\_matiere(matiere\_id)

Copy

Apply

app\routes\matiere\_routes.py

from flask import Blueprint, request, jsonify

from app.controllers.message\_controller import (

get\_messages\_received, get\_messages\_sent, send\_message,

mark\_as\_read, delete\_message

)

from functools import wraps

message\_bp = Blueprint('message', \_\_name\_\_)

# Middleware pour vérifier le format JSON

def require\_json(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if not request.is\_json:

return jsonify({"error": "Le format JSON est requis"}), 400

return f(\*args, \*\*kwargs)

return decorated\_function

# Routes pour les messages

@message\_bp.route('/users/<int:user\_id>/messages/received', methods=['GET'])

def received\_messages(user\_id):

return get\_messages\_received(user\_id)

@message\_bp.route('/users/<int:user\_id>/messages/sent', methods=['GET'])

def sent\_messages(user\_id):

return get\_messages\_sent(user\_id)

@message\_bp.route('/messages', methods=['POST'])

@require\_json

def create\_message():

return send\_message(request.json)

@message\_bp.route('/messages/<int:message\_id>/read', methods=['PUT'])

def read\_message(message\_id):

return mark\_as\_read(message\_id)

@message\_bp.route('/messages/<int:message\_id>', methods=['DELETE'])

def remove\_message(message\_id):

return delete\_message(message\_id)

Copy

Apply

app\routes\message\_routes.py

**4. Enregistrement des routes dans l'application principale**

Pour enregistrer toutes ces routes dans votre application Flask, vous devez mettre à jour votre fichier principal.

from flask import Flask

from flask\_sqlalchemy import SQLAlchemy

import os

# Initialisation de l'extension

db = SQLAlchemy()

def create\_app():

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

# Configuration

app.config['SQLALCHEMY\_DATABASE\_URI'] = os.environ.get('DATABASE\_URL', 'sqlite:///notes.db')

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

app.config['SECRET\_KEY'] = os.environ.get('SECRET\_KEY', 'dev\_key\_for\_development')

# Initialisation des extensions

db.init\_app(app)

# Enregistrement des blueprints

from app.routes.note\_routes import note\_bp

from app.routes.matiere\_routes import matiere\_bp

from app.routes.message\_routes import message\_bp

from app.routes.user\_routes import user\_bp

app.register\_blueprint(note\_bp, url\_prefix='/api')

app.register\_blueprint(matiere\_bp, url\_prefix='/api')

app.register\_blueprint(message\_bp, url\_prefix='/api')

app.register\_blueprint(user\_bp, url\_prefix='/api')

# Création des tables si elles n'existent pas

with app.app\_context():

db.create\_all()

return app

Copy

Apply

app\_\_init\_\_.py

from flask import Blueprint, request, jsonify

from app.controllers.user\_controller import login\_user, register\_user

from functools import wraps

user\_bp = Blueprint('user', \_\_name\_\_)

# Middleware pour vérifier le format JSON

def require\_json(f):

@wraps(f)

def decorated\_function(\*args, \*\*kwargs):

if not request.is\_json:

return jsonify({"error": "Le format JSON est requis"}), 400

return f(\*args, \*\*kwargs)

return decorated\_function

@user\_bp.route('/login', methods=['POST'])

@require\_json

def login():

return login\_user(request.json)

@user\_bp.route('/register', methods=['POST'])

@require\_json

def register():

return register\_user(request.json)

Copy

Apply

app\routes\user\_routes.py

**5. Fichier principal pour exécuter l'application**

from app import create\_app

app = create\_app()

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

app.run(debug=True)

Copy

Apply

run.py

**Documentation des endpoints API**

Voici un résumé des endpoints disponibles dans votre API:

**Utilisateurs**

* POST /api/login - Connexion d'un utilisateur
* POST /api/register - Inscription d'un nouvel utilisateur

**Matières**

* GET /api/matieres - Liste de toutes les matières
* GET /api/matieres/<id> - Détails d'une matière spécifique
* POST /api/matieres - Création d'une nouvelle matière
* PUT /api/matieres/<id> - Mise à jour d'une matière
* DELETE /api/matieres/<id> - Suppression d'une matière

**Notes**

* GET /api/etudiants/<id>/notes - Notes d'un étudiant
* GET /api/matieres/<id>/notes - Notes pour une matière
* POST /api/notes - Ajout d'une nouvelle note
* PUT /api/notes/<id> - Mise à jour d'une note
* DELETE /api/notes/<id> -