import cors from "cors";

import mysql from "mysql2";

import bcrypt from "bcryptjs";

import cookieParser from "cookie-parser";

import jwt from 'jsonwebtoken';

import bodyParser from 'body-parser';

import axios from 'axios';

import express from 'express';

import authRoutes from './routes/auth.js';

import connection from './db.js';

import XLSX from 'xlsx';

import { mkdirSync, existsSync } from 'fs';

import fs from "fs";

import dotenv from 'dotenv';

dotenv.config();

const app = express();

const router = express.Router();

//app.use('/api', router);

import multer from 'multer';

import path from 'path';

import { fileURLToPath } from 'url';

const \_\_filename = fileURLToPath(import.meta.url);

const \_\_dirname = path.dirname(\_\_filename);

// Configuration JWT améliorée

const JWT\_SECRET = process.env.JWT\_SECRET || 'dev-secret-only';

if (!process.env.JWT\_SECRET && process.env.NODE\_ENV === 'production') {

throw new Error('❌ Configuration critique: JWT\_SECRET doit être défini en production');

}

// Configuration JWT

const JWT\_CONFIG = {

secret: process.env.JWT\_SECRET || 'dev-secret-only',

expiresIn: '24h' // Durée de validité du token

};

// Vérification de la configuration JWT

if (!process.env.JWT\_SECRET) {

console.warn('⚠️ Avertissement: JWT\_SECRET non défini dans .env - utilisation d\'une clé de développement');

if (process.env.NODE\_ENV === 'production') {

throw new Error('Configuration critique: JWT\_SECRET doit être défini en production');

}

}

//const jwtSecret = process.env.JWT\_SECRET || 'dev-secret-only';

axios.defaults.baseURL = 'http://localhost:5000'; // Point direct vers le backend

axios.defaults.withCredentials = true;

// Middlewares

app.use(cors({

origin: 'http://localhost:3000',

credentials: true,

methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],

allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With']

}));

app.use(express.json());

app.options('\*', cors());

app.use(cookieParser());

app.use('/api', authRoutes); // plus de /api

app.use((req, res, next) => {

res.header("Access-Control-Allow-Origin", "http://localhost:3000");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

// Configuration de la base de données

const pool = mysql.createPool({

host: "localhost",

user: "root",

password: "",

database: "mon\_pfe",

waitForConnections: true,

connectionLimit: 10,

queueLimit: 0,

}).promise();

// Utilisez une URL absolue en développement

const API\_URL = process.env.NODE\_ENV === 'development'

? 'http://localhost:5000/api'

: '/api';

// Exemple d'appel corrigé

axios.get(`${API\_URL}/filieres`)

.then(response => console.log(response.data))

.catch(error => {

if (error.code === 'ECONNREFUSED') {

console.error('Serveur inaccessible - Vérifiez que le backend est démarré');

}

});

// Route pour la connexion admin

app.post("/admin/login", async (req, res) => {

const { email, password } = req.body;

try {

const [admin] = await pool.query("SELECT \* FROM admin WHERE Email = ?", [email]);

if (admin.length === 0) {

return res.status(401).json({ success: false, message: "Email incorrect" });

}

// Vérification simple car mot de passe non hashé dans votre DB

if (password !== admin[0].password) {

return res.status(401).json({ success: false, message: "Mot de passe incorrect" });

}

const token = jwt.sign(

{ email: admin[0].Email, role: 'admin' }, // Utilisez email plutôt que CIN

JWT\_CONFIG.secret,

{ expiresIn: JWT\_CONFIG.expiresIn }

);

res.json({

success: true,

token,

user: {

email: admin[0].Email,

role: 'admin'

}

});

} catch (err) {

console.error("Erreur de connexion admin:", err);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

// Route pour la connexion des utilisateurs

app.post("/connexion", async (req, res) => {

const { email, password } = req.body;

try {

// Vérification enseignant

const [enseignants] = await pool.query("SELECT \* FROM enseignants WHERE Email = ?", [email]);

if (enseignants.length > 0) {

const enseignant = enseignants[0];

const isMatch = await bcrypt.compare(password, enseignant.Password);

if (isMatch) {

const token = jwt.sign(

{ cin: enseignant.CIN, role: 'enseignant' },

process.env.JWT\_SECRET || 'dev-secret-only',

{ expiresIn: '24h' }

);

return res.json({

success: true,

message: "Connexion réussie",

token: token, // Important

role: "enseignant",

cin: enseignant.CIN,

email: enseignant.Email

});

}

}

// Vérification étudiant

const [etudiants] = await pool.query("SELECT \* FROM etudiant WHERE email = ?", [email]);

if (etudiants.length > 0) {

const etudiant = etudiants[0];

const isPasswordValid = await bcrypt.compare(password, etudiant.Password);

if (isPasswordValid) {

const token = jwt.sign(

{ cin: etudiant.CIN, role: 'etudiant' },

process.env.JWT\_SECRET || 'dev-secret-only',

{ expiresIn: '1h' }

);

return res.json({

success: true,

message: "Connexion réussie",

token: token,

role: "etudiant",

cin: etudiant.CIN,

email: etudiant.email

});

}

}

return res.status(401).json({

success: false,

message: "Identifiants incorrects"

});

} catch (error) {

console.error("Erreur lors de la connexion :", error);

return res.status(500).json({

success: false,

message: "Erreur interne du serveur"

});

}

});

app.post("/enseignants", async (req, res) => {

const {

Cin,

Nom\_et\_prénom,

Email,

Password,

Numero\_tel,

Classement,

Description

} = req.body;

// Validation des données

const errors = {};

const passwordFeedback = {

requirements: {

minLength: false,

hasUpper: false,

hasLower: false,

hasNumber: false,

hasSpecial: false

},

strength: 0

};

// Validation du CIN

if (!Cin) {

errors.Cin = "Le CIN est requis.";

} else if (!/^[01]\d{7}$/.test(Cin)) {

errors.Cin = "Le CIN doit contenir exactement 8 chiffres commençant par 0 ou 1.";

}

// Validation du nom

if (!Nom\_et\_prénom?.trim()) {

errors.Nom\_et\_prénom = "Le nom est requis.";

}

// Validation de l'email

if (!Email?.trim()) {

errors.email = "L'email est requis.";

} else if (!/\S+@\S+\.\S+/.test(Email)) {

errors.email = "L'email est invalide.";

}

// Validation du téléphone

if (!Numero\_tel) {

errors.Numero\_tel = "Le numéro de téléphone est requis.";

} else if (!/^\d{8}$/.test(Numero\_tel)) {

errors.Numero\_tel = "Le numéro doit contenir exactement 8 chiffres.";

}

// Validation du mot de passe

if (!Password) {

errors.password = "Le mot de passe est requis.";

} else {

passwordFeedback.requirements = {

minLength: Password.length >= 10,

hasUpper: /[A-Z]/.test(Password),

hasLower: /[a-z]/.test(Password),

hasNumber: /[0-9]/.test(Password),

hasSpecial: /[!@#$%^&\*(),.?":{}|<>]/.test(Password)

};

passwordFeedback.strength = Object.values(passwordFeedback.requirements).filter(Boolean).length;

if (passwordFeedback.strength < 3) {

errors.password = "Le mot de passe est trop faible.";

}

}

// Validation du classement

if (!Classement?.trim()) {

errors.Classement = "Le classement est requis.";

}

// Validation de la description

if (!Description?.trim()) {

errors.Description = "La description est requise.";

}

if (Object.keys(errors).length > 0) {

return res.status(400).json({

success: false,

errors,

passwordFeedback

});

}

try {

// Vérification des doublons

const [existing] = await pool.query(

"SELECT \* FROM enseignants WHERE Cin = ? OR Email = ?",

[Cin, Email]

);

if (existing.length > 0) {

return res.status(400).json({

success: false,

message: "Un enseignant avec ce CIN ou cet email existe déjà."

});

}

// Hashage du mot de passe

const saltRounds = 10;

const hashedPassword = await bcrypt.hash(Password, saltRounds);

// Insertion dans la base de données

await pool.query(

`INSERT INTO enseignants

(Cin, Nom\_et\_prénom, Email, Numero\_tel, Password, Classement, Description)

VALUES (?, ?, ?, ?, ?, ?, ?)`,

[

Cin,

Nom\_et\_prénom.trim(),

Email.trim(),

Numero\_tel,

hashedPassword,

Classement.trim(),

Description.trim()

]

);

res.status(201).json({

success: true,

message: "Inscription réussie",

data: { Cin, Nom\_et\_prénom, Email }

});

} catch (error) {

console.error("Erreur serveur:", error);

res.status(500).json({

success: false,

message: "Erreur lors de l'inscription",

error: process.env.NODE\_ENV === 'development' ? error.message : undefined

});

}

});

app.post("/etudiant", async (req, res) => {

const {

Cin,

Nom\_et\_prénom,

Téléphone,

email,

password,

confirmPassword,

filiere, // Notez le nom sans accent

classe

} = req.body;

// Validation de base

const errors = {};

if (!Cin) errors.Cin = "Le CIN est requis";

if (!Nom\_et\_prénom) errors.Nom\_et\_prénom = "Le nom est requis";

// Ajoutez les autres validations...

if (Object.keys(errors).length > 0) {

return res.status(400).json({

success: false,

errors

});

}

try {

// Vérifiez si l'étudiant existe déjà

const [existing] = await pool.query(

"SELECT \* FROM etudiant WHERE Cin = ? OR email = ?",

[Cin, email]

);

if (existing.length > 0) {

return res.status(400).json({

success: false,

message: "Un étudiant avec ce CIN ou cet email existe déjà"

});

}

// Hashage du mot de passe

const hashedPassword = await bcrypt.hash(password, 10);

const hashedConfirmPassword = await bcrypt.hash(confirmPassword, 10);

// Insertion dans la base

await pool.query(

`INSERT INTO etudiant

(Cin, Nom\_et\_prénom, Téléphone, email, password, Confirmpassword, Filière, Classe)

VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,

[Cin, Nom\_et\_prénom, Téléphone, email, hashedPassword, hashedConfirmPassword, filiere, classe]

);

res.status(201).json({

success: true,

message: "Inscription réussie"

});

} catch (error) {

console.error("Erreur serveur:", error);

res.status(500).json({

success: false,

message: "Erreur serveur"

});

}

});

// Route pour afficher tous les utilisateurs

app.get("/api/utilisateurs", async (req, res) => {

try {

const [rows] = await pool.query("SELECT \* FROM vue\_utilisateurs");

res.json(rows);

} catch (err) {

console.error("Erreur SQL :", err);

res.status(500).json({ error: "Erreur serveur" });

}

});

// Route pour supprimer un utilisateur

app.delete('/api/utilisateurs/:cin', async (req, res) => {

const { cin } = req.params;

const { role } = req.query;

try {

if (role === 'enseignant') {

await pool.query('DELETE FROM enseignants WHERE Cin = ?', [cin]);

} else if (role === 'étudiant') {

await pool.query('DELETE FROM etudiant WHERE Cin = ?', [cin]);

} else {

return res.status(400).json({ message: 'Rôle non valide' });

}

res.status(200).json({ message: 'Utilisateur supprimé avec succès' });

} catch (error) {

console.error('Erreur suppression :', error);

res.status(500).json({ message: 'Erreur serveur', error: error.message });

}

});

// Route pour modifier un utilisateur

app.put("/api/utilisateurs/:cin", async (req, res) => {

const { cin } = req.params;

const { nom, email, formation, role } = req.body;

try {

if (role === "enseignant") {

await pool.query(

"UPDATE enseignants SET Nom\_et\_prénom = ?, Email = ? WHERE CIN = ?",

[nom, email, cin]

);

} else if (role === "étudiant") {

await pool.query(

"UPDATE etudiant SET Nom\_et\_prénom = ?, email = ?, filière = ? WHERE CIN = ?",

[nom, email, formation, cin]

);

}

res.json({ message: "Utilisateur mis à jour" });

} catch (err) {

console.error("Erreur modification :", err);

res.status(500).json({ error: "Erreur serveur" });

}

});

// Route pour enregistrer un participant

// Route pour enregistrer un participant

app.post('/api/register', async (req, res) => {

console.log('Données reçues:', req.body);

try {

const { nom, cin, email, tele, sexe, niveauEtude } = req.body;

// Validation améliorée

const errors = [];

if (!nom?.trim()) errors.push('Le nom est requis');

if (!cin) errors.push('Le CIN est requis');

if (!email?.trim()) errors.push('L\'email est requis');

if (!tele) errors.push('Le téléphone est requis');

if (!sexe) errors.push('Le sexe est requis');

if (!niveauEtude) errors.push('Le niveau d\'étude est requis');

if (errors.length > 0) {

return res.status(400).json({

success: false,

message: 'Validation échouée',

errors

});

}

// Conversion des types

const connection = await pool.getConnection();

try {

const [result] = await connection.execute(

'INSERT INTO formulaire (Nom\_complet, CIN, Email, Numéro\_téléphone, Sexe, Niveau\_étude) VALUES (?, ?, ?, ?, ?, ?)',

[

nom.trim(),

Number(cin),

email.trim(),

Number(tele),

sexe,

niveauEtude

]

);

res.status(201).json({

success: true,

message: 'Inscription réussie',

id: result.insertId

});

} finally {

connection.release();

}

} catch (error) {

console.error('Erreur serveur:', error);

let errorMessage = 'Erreur lors de l\'inscription';

if (error.code === 'ER\_DUP\_ENTRY') {

errorMessage = 'Ce CIN ou cet email est déjà enregistré';

} else if (error.code === 'ER\_TRUNCATED\_WRONG\_VALUE\_FOR\_FIELD') {

errorMessage = 'Format de données incorrect pour un champ numérique';

}

res.status(500).json({

success: false,

message: errorMessage,

error: process.env.NODE\_ENV === 'development' ? error.message : undefined

});

}

});

// Middleware d'authentification amélioré

// Middleware d'authentification générique

// Middleware d'authentification générique

const authenticate = (allowedRoles) => async (req, res, next) => {

const authHeader = req.headers.authorization;

if (!authHeader?.startsWith('Bearer ')) {

return res.status(401).json({ success: false, message: "Token manquant" });

}

const token = authHeader.split(' ')[1];

try {

const decoded = jwt.verify(token, JWT\_SECRET);

// Vérification du rôle

if (!allowedRoles.includes(decoded.role)) {

return res.status(403).json({

success: false,

message: "Accès non autorisé pour ce rôle"

});

}

// Vérification en base de données

let user;

switch(decoded.role) {

case 'admin':

[user] = await pool.query('SELECT \* FROM admin WHERE Email = ?', [decoded.email]);

break;

case 'enseignant':

[user] = await pool.query('SELECT \* FROM enseignants WHERE CIN = ?', [decoded.cin]);

break;

case 'etudiant':

[user] = await pool.query('SELECT \* FROM etudiant WHERE CIN = ?', [decoded.cin]);

break;

case 'Agent':

case 'Superviseur':

case 'Administrateur':

[user] = await pool.query('SELECT \* FROM agents WHERE id = ?', [decoded.id]);

break;

default:

throw new Error('Rôle invalide');

}

if (!user.length) throw new Error('Utilisateur non trouvé');

req.user = { ...decoded, details: user[0] };

next();

} catch (error) {

console.error(`Erreur d'authentification: ${error.message}`);

res.status(401).json({

success: false,

message: error.message.includes('jwt expired')

? 'Session expirée'

: 'Authentification échouée'

});

}

};

// Middleware spécifique pour les agents

const authenticateAgent = (allowedRoles = ['Agent', 'Superviseur', 'Administrateur']) => {

return authenticate(allowedRoles);

};

// Middlewares spécifiques par rôle

const authenticateAdmin = authenticate(['admin']);

const authenticateTeacher = authenticate(['enseignant']);

const authenticateStudent = authenticate(['etudiant']);

app.get("/api/enseignants", authenticate(['enseignant']), async (req, res) => {

try {

const { cin, email } = req.query;

if (!cin && !email) {

return res.status(400).json({

success: false,

message: "Paramètres manquants: cin ou email requis"

});

}

const [results] = await pool.query(

`SELECT

Cin, Nom\_et\_prénom, Email,

Numero\_tel, Classement, Description,

ProfileImage

FROM enseignants

WHERE Cin = ? OR Email = ?

LIMIT 1`,

[cin || null, email || null]

);

if (results.length === 0) {

return res.status(404).json({

success: false,

message: "Enseignant non trouvé"

});

}

const teacherData = results[0];

res.json({

success: true,

data: {

...results[0],

// Assurez-vous que le chemin est complet

ProfileImage: results[0].ProfileImage

? `http://localhost:5000${results[0].ProfileImage}`

: null

}

});

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({

success: false,

message: "Erreur serveur",

error: process.env.NODE\_ENV === 'development' ? error.message : undefined

});

}

});

const storage = multer.diskStorage({

destination: (req, file, cb) => {

cb(null, uploadDir);

},

filename: (req, file, cb) => {

const ext = path.extname(file.originalname);

const uniqueName = `profile-${Date.now()}${ext}`;

cb(null, uniqueName);

}

});

const upload = multer({

storage: storage,

limits: { fileSize: 2 \* 1024 \* 1024 }, // 2MB

fileFilter: (req, file, cb) => {

if (file.mimetype.match(/^image\/(jpeg|png|jpg)$/)) {

cb(null, true);

} else {

cb(new Error('Type de fichier non supporté'), false);

}

}

});

// 1. Configuration du stockage des images

const profileStorage = multer.diskStorage({

destination: (req, file, cb) => {

const dir = path.join(\_\_dirname, 'uploads/profiles');

if (!fs.existsSync(dir)) fs.mkdirSync(dir, { recursive: true }); // Crée le dossier si inexistant

cb(null, dir);

},

filename: (req, file, cb) => {

const ext = path.extname(file.originalname);

const uniqueName = `profile-${Date.now()}${ext}`;

cb(null, uniqueName); // Ex: "profile-1623456789.png"

}

});

// 2. Middleware Multer

const uploadProfile = multer({

storage: profileStorage,

limits: { fileSize: 2 \* 1024 \* 1024 }, // 2MB max

fileFilter: (req, file, cb) => {

if (file.mimetype.match(/^image\/(jpeg|png|jpg)$/)) {

cb(null, true); // Accepte l'image

} else {

cb(new Error('Seules les images (JPEG/PNG) sont autorisées !'), false);

}

}

});

// Configuration for schedule files (emploi du temps)

const scheduleStorage = multer.diskStorage({

destination: (req, file, cb) => {

const uploadDir = path.join(\_\_dirname, "uploads", "emplois");

if (!fs.existsSync(uploadDir)) {

fs.mkdirSync(uploadDir, { recursive: true });

}

cb(null, uploadDir);

},

filename: (req, file, cb) => {

const ext = path.extname(file.originalname);

const uniqueName = `emploi-${Date.now()}${ext}`;

cb(null, uniqueName);

}

});

const uploadSchedule = multer({

storage: scheduleStorage,

limits: { fileSize: 5 \* 1024 \* 1024 }, // 5MB

fileFilter: (req, file, cb) => {

const allowedTypes = [

'application/pdf',

'application/vnd.ms-excel',

'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'

];

if (allowedTypes.includes(file.mimetype)) {

cb(null, true);

} else {

cb(new Error('Seuls les fichiers PDF et Excel sont autorisés'), false);

}

}

});

const uploadDirs = [

path.join(\_\_dirname, 'uploads', 'profiles'),

path.join(\_\_dirname, 'uploads', 'documents'),

path.join(\_\_dirname, 'uploads', 'emplois')

];

uploadDirs.forEach(dir => {

if (!fs.existsSync(dir)) {

fs.mkdirSync(dir, { recursive: true });

}

});

app.use('/api/uploads', express.static(path.join(\_\_dirname, 'uploads')));

{/\*app.get('/api/emplois/:id/download', async (req, res) => {

try {

const [emploi] = await pool.query(

'SELECT fichier\_path FROM emplois\_du\_temps WHERE id = ?',

[req.params.id]

);

if (!emploi[0]?.fichier\_path) {

return res.status(404).json({ error: 'Fichier non trouvé' });

}

const filePath = path.join(\_\_dirname, emploi[0].fichier\_path);

res.download(filePath);

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({ error: 'Erreur serveur' });

}

});\*/}

// Emploi du temps enseignant publié + parsing

app.get('/api/emplois/enseignant/:cin', async (req, res) => {

try {

const { cin } = req.params;

const [emplois] = await pool.query(`

SELECT \*

FROM emplois\_du\_temps

WHERE enseignant\_id = ? AND type = 'enseignant' AND published = 1

ORDER BY created\_at DESC

LIMIT 1

`, [cin]);

res.json({ success: true, data: emplois });

} catch (err) {

console.error('Erreur emploi enseignant:', err);

res.status(500).json({ success: false, message: 'Erreur serveur' });

}

});

app.get('/api/emplois/:id/parsed', async (req, res) => {

try {

const [emploi] = await pool.query(

'SELECT fichier\_path FROM emplois\_du\_temps WHERE id = ?',

[req.params.id]

);

if (!emploi[0]?.fichier\_path) {

return res.status(404).json({ success: false, message: 'Fichier non trouvé' });

}

const filePath = path.join(\_\_dirname, emploi[0].fichier\_path);

const workbook = XLSX.readFile(filePath);

const sheet = workbook.Sheets[workbook.SheetNames[0]];

const data = XLSX.utils.sheet\_to\_json(sheet, { header: 1 });

const headers = data[0];

const rows = data.slice(1).map(row =>

Object.fromEntries(headers.map((h, i) => [h, row[i]]))

);

res.json({ success: true, data: { headers, rows } });

} catch (error) {

console.error("Erreur parsing:", error);

res.status(500).json({ success: false, message: 'Erreur serveur' });

}

});

// Ajouter un nouvel emploi du temps

app.post("/api/emplois", uploadSchedule.single("fichier"), async (req, res) => {

try {

const { type, enseignant\_id } = req.body;

const fichier\_path = req.file ? `/uploads/emplois/${req.file.filename}` : null;

// Validation spécifique au type

if (type === 'enseignant') {

if (!enseignant\_id || !fichier\_path) {

return res.status(400).json({

success: false,

message: "Pour un emploi enseignant, le fichier et l'enseignant sont requis"

});

}

} else {

const { filiere\_id, classe\_id, semestre\_id } = req.body;

if (!filiere\_id || !classe\_id || !semestre\_id || !fichier\_path) {

return res.status(400).json({

success: false,

message: "Pour un emploi étudiant, tous les champs sont requis"

});

}

}

// Requête SQL conditionnelle

const [result] = await pool.query(

`INSERT INTO emplois\_du\_temps

(type, fichier\_path, enseignant\_id, filiere\_id, classe\_id, semestre\_id)

VALUES (?, ?, ?, ?, ?, ?)`,

[

type,

fichier\_path,

type === 'enseignant' ? enseignant\_id : null,

type === 'etudiant' ? req.body.filiere\_id : null,

type === 'etudiant' ? req.body.classe\_id : null,

type === 'etudiant' ? req.body.semestre\_id : null

]

);

res.status(201).json({ success: true, id: result.insertId });

} catch (error) {

console.error("Erreur SQL:", error);

res.status(500).json({

success: false,

message: error.sqlMessage || "Erreur serveur"

});

}

});

// Récupérer les emplois publiés par classe et type

app.get("/api/emplois/classe/:classeNom", async (req, res) => {

try {

const { classeNom } = req.params;

const { type } = req.query;

const [emplois] = await pool.query(`

SELECT e.\*,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero,

DATE\_FORMAT(e.published\_at, '%Y-%m-%d %H:%i:%s') AS published\_at

FROM emplois\_du\_temps e

LEFT JOIN filieres f ON e.filiere\_id = f.id

LEFT JOIN classes c ON e.classe\_id = c.id

LEFT JOIN semestres s ON e.semestre\_id = s.id

WHERE e.published = TRUE

AND c.nom = ?

AND e.type = ?

ORDER BY e.created\_at DESC

`, [classeNom, type]);

res.json({

success: true,

data: emplois,

});

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({

success: false,

message: "Erreur serveur",

});

}

});

// Publier un emploi du temps

// Publier un emploi du temps

// Route PUT pour publier un emploi du temps

app.put("/api/emplois/:id/publish", async (req, res) => {

try {

const [result] = await pool.query(

`UPDATE emplois\_du\_temps

SET published = 1,

published\_at = NOW()

WHERE id = ?`,

[req.params.id]

);

if (result.affectedRows === 0) {

return res.status(404).json({

success: false,

message: "Emploi du temps non trouvé"

});

}

res.json({

success: true,

message: "Emploi du temps publié avec succès"

});

} catch (error) {

console.error("Erreur lors de la publication:", error);

res.status(500).json({

success: false,

message: "Erreur serveur"

});

}

});

// Télécharger un emploi du temps

app.get("/api/emplois/:id/download", async (req, res) => {

try {

const { id } = req.params;

const [emploi] = await pool.query(

"SELECT fichier\_path FROM emplois\_du\_temps WHERE id = ?",

[id]

);

if (!emploi.length) {

return res.status(404).json({ message: "Emploi non trouvé" });

}

const filePath = path.join(\_\_dirname, emploi[0].fichier\_path);

if (!fs.existsSync(filePath)) {

return res.status(404).json({ message: "Fichier introuvable sur le serveur" });

}

res.download(filePath, err => {

if (err) console.error("Erreur de téléchargement:", err);

});

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({ message: "Erreur serveur" });

}

});

// Routes pour les données de référence

app.get("/api/filieres", async (req, res) => {

try {

const [filieres] = await pool.query("SELECT \* FROM filieres ORDER BY nom");

res.json({

success: true,

data: filieres,

});

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({

success: false,

message: "Erreur serveur",

});

}

});

app.get("/api/classes", async (req, res) => {

try {

const [classes] = await pool.query("SELECT \* FROM classes ORDER BY nom");

res.json({

success: true,

data: classes,

});

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({

success: false,

message: "Erreur serveur",

});

}

});

// Dans votre backend (server.js)

// Dans votre backend (server.js)

app.get('/api/semestres', async (req, res) => {

try {

const [semestres] = await pool.query(`

SELECT s.id, s.numero, c.nom AS classe\_nom, f.nom AS filiere\_nom

FROM semestres s

JOIN classes c ON s.classe\_id = c.id

JOIN filieres f ON c.filiere\_id = f.id

ORDER BY s.numero

`);

res.json({ success: true, data: semestres });

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

// Middleware pour les erreurs

app.use((err, req, res, next) => {

console.error(err.stack);

res.status(500).json({

success: false,

message: 'Erreur interne du serveur',

error: process.env.NODE\_ENV === 'development' ? err.message : undefined

});

});

// Route pour l'emploi du temps de l'étudiant

app.get("/api/etudiant/:cin/emploi-du-temps", async (req, res) => {

// Implémentation similaire à la route précédente

});

app.get('/api/teachers/profile', authenticate(['enseignant']), async (req, res) => {

try {

const [results] = await pool.query(

'SELECT CIN, Nom\_et\_prénom, Email, Numero\_tel, Classement, Description, profile\_image FROM enseignants WHERE CIN = ?',

[req.user.cin]

);

if (results.length === 0) {

return res.status(404).json({ message: 'Enseignant non trouvé' });

}

res.json(results[0]);

} catch (error) {

console.error('Error:', error);

res.status(500).json({ message: 'Erreur serveur' });

}

});

app.use('/uploads', express.static(path.join(\_\_dirname, 'uploads')));

// Route pour l'upload

app.post('/api/teachers/upload-profile', upload.single('profile'), async (req, res) => {

try {

if (!req.file) {

return res.status(400).json({ success: false, message: 'Aucun fichier téléchargé' });

}

const token = req.headers.authorization?.split(' ')[1];

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const imagePath = `/uploads/profiles/${req.file.filename}`;

await pool.query(

'UPDATE enseignants SET profile\_image = ? WHERE CIN = ?',

[imagePath, decoded.cin]

);

res.json({

success: true,

imageUrl: imagePath

});

} catch (error) {

console.error('Upload error:', error);

res.status(500).json({ success: false, message: 'Erreur serveur' });

}

});

// Route pour l'inscription des étudiants

app.post('/api/etudiant', (req, res) => {

// Traitement de l'inscription

console.log(req.body);

res.status(201).json({ message: "Inscription réussie" });

});

// Route pour récupérer les données d'un étudiant

// Dans server.js (backend)

app.get('/api/etudiant/:cin', async (req, res) => {

try {

const { cin } = req.params;

const [etudiant] = await pool.query(

'SELECT CIN, Nom\_et\_prénom, Téléphone, Email, Filière, Classe, ProfileImage FROM etudiant WHERE CIN = ?',

[cin]

);

if (!etudiant.length) {

return res.status(404).json({ success: false, message: "Étudiant non trouvé" });

}

res.json({

success: true,

data: {

...etudiant[0],

// Assurez-vous que ProfileImage est bien renvoyé même s'il est NULL

ProfileImage: etudiant[0].ProfileImage || null

}

});

} catch (error) {

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

// Route pour l'upload des photos étudiants

app.post('/api/etudiant/upload-profile',

authenticate(['etudiant']), // Middleware d'authentification

uploadProfile.single('profile'), // 'profile' = nom du champ dans FormData

async (req, res) => {

try {

if (!req.file) {

return res.status(400).json({ success: false, message: "Aucun fichier reçu." });

}

const token = req.headers.authorization.split(' ')[1];

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

const cin = decoded.cin;

// Chemin relatif (ex: "/uploads/profiles/profile-123456789.png")

const imagePath = `/uploads/profiles/${req.file.filename}`;

// Mise à jour en base de données

await pool.query(

'UPDATE etudiant SET ProfileImage = ? WHERE CIN = ?',

[imagePath, cin]

);

res.json({

success: true,

imageUrl: imagePath // Renvoie le chemin pour affichage immédiat

});

} catch (error) {

console.error("Erreur upload:", error);

res.status(500).json({

success: false,

message: "Erreur serveur lors de l'enregistrement de la photo.",

error: process.env.NODE\_ENV === 'development' ? error.message : undefined

});

}

}

);

// Middleware essentiels

app.use(bodyParser.json()); // Pour parser le JSON

app.use(bodyParser.urlencoded({ extended: true })); // Pour parser les formulaires

////////filière

app.post('/api/filieres', async (req, res) => {

try {

const { nom } = req.body;

if (!nom) {

return res.status(400).json({

success: false,

message: 'Le nom de la filière est requis'

});

}

const [result] = await pool.execute(

'INSERT INTO filieres (nom) VALUES (?)',

[nom]

);

res.status(201).json({

success: true,

id: result.insertId,

nom,

message: 'Filière créée avec succès'

});

} catch (error) {

console.error('Erreur lors de la création de la filière:', error);

// Gestion spécifique des erreurs de doublon

if (error.code === 'ER\_DUP\_ENTRY') {

return res.status(400).json({

success: false,

message: 'Une filière avec ce nom existe déjà'

});

}

res.status(500).json({

success: false,

message: 'Erreur serveur lors de la création de la filière',

error: process.env.NODE\_ENV === 'development' ? error.message : undefined

});

}

});

;

app.put('/api/filieres/:id', async (req, res) => {

try {

const { id } = req.params;

const { nom } = req.body;

if (!nom) {

return res.status(400).json({ error: 'Le nom de la filière est requis' });

}

const [result] = await pool.execute(

'UPDATE filieres SET nom = ? WHERE id = ?',

[nom, id]

);

if (result.affectedRows === 0) {

return res.status(404).json({ error: 'Filière non trouvée' });

}

res.json({ id, nom, message: 'Filière mise à jour avec succès' });

} catch (error) {

console.error('Erreur lors de la mise à jour de la filière:', error);

res.status(500).json({ error: 'Erreur serveur' });

}

});

app.delete('/api/filieres/:id', async (req, res) => {

try {

const { id } = req.params;

const [result] = await pool.execute(

'DELETE FROM filieres WHERE id = ?',

[id]

);

if (result.affectedRows === 0) {

return res.status(404).json({ error: 'Filière non trouvée' });

}

res.json({ message: 'Filière supprimée avec succès' });

} catch (error) {

console.error('Erreur lors de la suppression de la filière:', error);

// Gestion spécifique des contraintes de clé étrangère

if (error.code === 'ER\_ROW\_IS\_REFERENCED\_2') {

return res.status(400).json({

error: 'Impossible de supprimer : des classes sont associées à cette filière'

});

}

res.status(500).json({ error: 'Erreur serveur' });

}

});

/////classes

// Routes pour les classes

// Route POST pour ajouter une classe

app.post('/api/classes', async (req, res) => {

try {

const { nom, filiere\_id } = req.body;

// Validation

if (!nom || !filiere\_id) {

return res.status(400).json({

success: false,

error: 'Le nom et l\'ID de la filière sont requis'

});

}

// Insertion dans la base de données

const [result] = await pool.query(

'INSERT INTO classes (nom, filiere\_id) VALUES (?, ?)',

[nom, filiere\_id]

);

res.status(201).json({

success: true,

message: 'Classe créée avec succès',

data: {

id: result.insertId,

nom,

filiere\_id

}

});

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({

success: false,

error: 'Erreur serveur'

});

}

});

// Testez la connexion au démarrage

pool.getConnection((err, connection) => {

if (err) {

console.error('Erreur de connexion à la base de données:', err);

} else {

console.log('Connecté à la base de données MySQL');

connection.release();

}

});

// Récupérer toutes les classes

app.get('/api/classes', async (req, res) => {

try {

const [classes] = await pool.query(`

SELECT c.id, c.nom, f.nom as filiere\_nom, f.id as filiere\_id

FROM classes c

JOIN filieres f ON c.filiere\_id = f.id

`);

res.json(classes);

} catch (err) {

console.error(err);

res.status(500).json({ error: 'Erreur serveur' });

}

});

// Dans server.js ou votre fichier de routes

app.put('/api/classes/:id', async (req, res) => {

try {

const { id } = req.params;

const { nom, filiere\_id } = req.body;

// Validation

if (!nom || !filiere\_id) {

return res.status(400).json({ error: 'Nom et filière sont requis' });

}

// Mise à jour dans la base de données

const [result] = await pool.query(

'UPDATE classes SET nom = ?, filiere\_id = ? WHERE id = ?',

[nom, filiere\_id, id]

);

if (result.affectedRows === 0) {

return res.status(404).json({ error: 'Classe non trouvée' });

}

res.json({ success: true, message: 'Classe mise à jour' });

} catch (error) {

console.error('Erreur modification classe:', error);

res.status(500).json({ error: 'Erreur serveur' });

}

});

// Route DELETE pour supprimer une classe

// Route DELETE pour supprimer une classe

app.delete('/api/classes/:id', async (req, res) => {

try {

const { id } = req.params;

// 1. Vérifier si la classe existe

const [classe] = await pool.query('SELECT id FROM classes WHERE id = ?', [id]);

if (!classe.length) {

return res.status(404).json({

success: false,

message: 'Classe non trouvée'

});

}

// 2. Suppression effective

await pool.query('DELETE FROM classes WHERE id = ?', [id]);

res.json({

success: true,

message: 'Classe supprimée avec succès'

});

} catch (error) {

console.error('Erreur suppression:', error);

// Gestion des contraintes de clé étrangère

if (error.code === 'ER\_ROW\_IS\_REFERENCED\_2') {

return res.status(400).json({

success: false,

message: 'Impossible de supprimer : des éléments sont liés à cette classe'

});

}

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Récupérer les classes d'une filière spécifique

app.get('/api/filieres/:filiereId/classes', async (req, res) => {

try {

const [classes] = await pool.query(

'SELECT id, nom FROM classes WHERE filiere\_id = ?',

[req.params.filiereId]

);

res.json({

success: true,

data: classes

});

} catch (error) {

console.error('Error fetching classes:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

app.get('/api/classes/:classId/semestres', async (req, res) => {

try {

const [semestres] = await pool.query(

'SELECT id, numero FROM semestres WHERE classe\_id = ?',

[req.params.classId]

);

res.json({

success: true,

data: semestres

});

} catch (error) {

console.error('Error fetching semesters:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

app.get('/api/semestres/:semesterId/matieres', async (req, res) => {

try {

const [matieres] = await pool.query(

'SELECT id, nom FROM matieres WHERE semestre\_id = ?',

[req.params.semesterId]

);

res.json({

success: true,

data: matieres

});

} catch (error) {

console.error('Error fetching subjects:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

app.get('/api/classes/:classeId/matieres', async (req, res) => {

try {

const [matieres] = await pool.query(`

SELECT m.id, m.nom

FROM matieres m

JOIN semestres s ON m.semestre\_id = s.id

WHERE s.classe\_id = ?

`, [req.params.classeId]);

res.json(matieres);

} catch (error) {

res.status(500).json({ error: error.message });

}

});

///semestres

// Route POST pour créer un semestre

// Route POST pour créer un semestre

app.post('/api/semestres', async (req, res) => {

try {

const { numero, classe\_id } = req.body;

// Validation

if (!numero || !classe\_id) {

return res.status(400).json({

success: false,

message: 'Le numéro et la classe sont obligatoires'

});

}

// Vérifier si la classe existe

const [classe] = await pool.query(

'SELECT id FROM classes WHERE id = ?',

[classe\_id]

);

if (!classe.length) {

return res.status(400).json({

success: false,

message: 'Classe non trouvée'

});

}

// Création du semestre

const [result] = await pool.query(

'INSERT INTO semestres (numero, classe\_id) VALUES (?, ?)',

[numero, classe\_id]

);

res.status(201).json({

success: true,

data: {

id: result.insertId,

numero,

classe\_id

}

});

} catch (error) {

console.error('Erreur création semestre:', error);

if (error.code === 'ER\_DUP\_ENTRY') {

return res.status(400).json({

success: false,

message: 'Ce semestre existe déjà pour cette classe'

});

}

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Récupérer tous les semestres avec leur classe

{/\*app.get('/api/semestres', async (req, res) => {

try {

const [semestres] = await pool.query(`

SELECT s.id, s.numero, c.nom as classe\_nom, c.id as classe\_id

FROM semestres s

JOIN classes c ON s.classe\_id = c.id

`);

res.json(semestres);

} catch (err) {

console.error(err);

res.status(500).json({ error: 'Erreur serveur' });

}

});\*/}

// Récupérer les semestres d'une classe spécifique

app.get('/api/classes/:classeId/semestres', async (req, res) => {

try {

const [semestres] = await pool.query(

'SELECT \* FROM semestres WHERE classe\_id = ?',

[req.params.classeId]

);

res.json(semestres);

} catch (err) {

console.error(err);

res.status(500).json({ error: 'Erreur serveur' });

}

});

// Route DELETE pour supprimer un semestre

// Route DELETE pour supprimer un semestre

app.delete('/api/semestres/:id', async (req, res) => {

try {

const { id } = req.params;

// 1. Vérifier si le semestre existe

const [semestre] = await pool.query('SELECT id FROM semestres WHERE id = ?', [id]);

if (!semestre.length) {

return res.status(404).json({ success: false, message: "Semestre non trouvé" });

}

// 2. Supprimer le semestre

await pool.query('DELETE FROM semestres WHERE id = ?', [id]);

res.json({ success: true, message: "Semestre supprimé avec succès" });

} catch (error) {

console.error("Erreur lors de la suppression :", error);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

//////matieres

// Route POST pour créer une matière

// Route POST pour créer une matière

app.post('/api/matieres', async (req, res) => {

try {

const { nom, credits, enseignant\_id, semestre\_id } = req.body;

// Validation

if (!nom || !semestre\_id) {

return res.status(400).json({

success: false,

message: 'Le nom et le semestre sont obligatoires'

});

}

// Vérifier si le semestre existe

const [semestre] = await pool.query(

'SELECT id FROM semestres WHERE id = ?',

[semestre\_id]

);

if (!semestre.length) {

return res.status(400).json({

success: false,

message: 'Semestre non trouvé'

});

}

// Vérifier si l'enseignant existe (si renseigné)

if (enseignant\_id) {

const [enseignant] = await pool.query(

'SELECT Cin FROM enseignants WHERE Cin = ?',

[enseignant\_id]

);

if (!enseignant.length) {

return res.status(400).json({

success: false,

message: 'Enseignant non trouvé'

});

}

}

// Insertion

const [result] = await pool.query(

'INSERT INTO matieres (nom, credits, enseignant\_id, semestre\_id) VALUES (?, ?, ?, ?)',

[nom, credits, enseignant\_id, semestre\_id]

);

res.status(201).json({

success: true,

data: {

id: result.insertId,

...req.body

}

});

} catch (error) {

console.error('Erreur création matière:', error);

if (error.code === 'ER\_DUP\_ENTRY') {

return res.status(400).json({

success: false,

message: 'Cette matière existe déjà pour ce semestre'

});

}

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Route GET pour récupérer toutes les matières

app.get('/api/matieres', async (req, res) => {

try {

const [matieres] = await pool.query(`

SELECT m.id, m.nom, m.credits,

e.Nom\_et\_prénom AS enseignant,

s.numero AS semestre,

c.nom AS classe

FROM matieres m

LEFT JOIN enseignants e ON m.enseignant\_id = e.Cin

LEFT JOIN semestres s ON m.semestre\_id = s.id

LEFT JOIN classes c ON s.classe\_id = c.id

`);

res.json({

success: true,

data: matieres

});

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({

success: false,

error: 'Erreur serveur'

});

}

});

// Route PUT pour modifier une matière

app.put('/api/matieres/:id', async (req, res) => {

try {

const { id } = req.params;

const { nom, credits, enseignant\_id, semestre\_id } = req.body;

// Validation

if (!nom || !semestre\_id) {

return res.status(400).json({

success: false,

message: 'Le nom et le semestre sont obligatoires'

});

}

// Vérifier si la matière existe

const [matiere] = await pool.query(

'SELECT id FROM matieres WHERE id = ?',

[id]

);

if (!matiere.length) {

return res.status(404).json({

success: false,

message: 'Matière non trouvée'

});

}

// Vérifier le semestre

const [semestre] = await pool.query(

'SELECT id FROM semestres WHERE id = ?',

[semestre\_id]

);

if (!semestre.length) {

return res.status(400).json({

success: false,

message: 'Semestre non trouvé'

});

}

// Vérifier l'enseignant si renseigné

if (enseignant\_id) {

const [enseignant] = await pool.query(

'SELECT Cin FROM enseignants WHERE Cin = ?',

[enseignant\_id]

);

if (!enseignant.length) {

return res.status(400).json({

success: false,

message: 'Enseignant non trouvé'

});

}

}

// Mise à jour

await pool.query(

'UPDATE matieres SET nom = ?, credits = ?, enseignant\_id = ?, semestre\_id = ? WHERE id = ?',

[nom, credits, enseignant\_id, semestre\_id, id]

);

res.json({

success: true,

message: 'Matière mise à jour avec succès'

});

} catch (error) {

console.error('Erreur modification matière:', error);

if (error.code === 'ER\_DUP\_ENTRY') {

return res.status(400).json({

success: false,

message: 'Cette matière existe déjà pour ce semestre'

});

}

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Route DELETE pour supprimer une matière

app.delete('/api/matieres/:id', async (req, res) => {

try {

const { id } = req.params;

// 1. Vérifier si la matière existe

const [matiere] = await pool.query(

'SELECT id FROM matieres WHERE id = ?',

[id]

);

if (!matiere.length) {

return res.status(404).json({

success: false,

message: 'Matière non trouvée'

});

}

// 2. Suppression effective

await pool.query(

'DELETE FROM matieres WHERE id = ?',

[id]

);

res.json({

success: true,

message: 'Matière supprimée avec succès'

});

} catch (error) {

console.error('Erreur suppression matière:', error);

// Gestion des contraintes de clé étrangère

if (error.code === 'ER\_ROW\_IS\_REFERENCED\_2') {

return res.status(400).json({

success: false,

message: 'Impossible de supprimer : des éléments sont liés à cette matière'

});

}

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Route POST pour créer un événement

app.post('/api/evenements', async (req, res) => {

try {

const { titre, date, lieu, type, description } = req.body;

// Validation

if (!titre || !date || !lieu || !type) {

return res.status(400).json({

success: false,

message: 'Les champs titre, date, lieu et type sont obligatoires'

});

}

const [result] = await pool.query(

'INSERT INTO evenements (titre, date, lieu, type, description) VALUES (?, ?, ?, ?, ?)',

[titre, new Date(date), lieu, type, description || null]

);

res.status(201).json({

success: true,

data: {

id: result.insertId,

...req.body

}

});

} catch (error) {

console.error('Erreur création événement:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Route GET pour récupérer tous les événements

app.get('/api/evenements', async (req, res) => {

try {

const [evenements] = await pool.query(

'SELECT id, titre, date, lieu, type, description FROM evenements ORDER BY date DESC'

);

res.json({

success: true,

data: evenements

});

} catch (error) {

console.error('Erreur récupération événements:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Route PUT pour modifier un événement

app.put('/api/evenements/:id', async (req, res) => {

try {

const { id } = req.params;

const { titre, date, lieu, type, description } = req.body;

// Validation

if (!titre || !date || !lieu || !type) {

return res.status(400).json({

success: false,

message: 'Les champs titre, date, lieu et type sont obligatoires'

});

}

const [result] = await pool.query(

'UPDATE evenements SET titre = ?, date = ?, lieu = ?, type = ?, description = ? WHERE id = ?',

[titre, new Date(date), lieu, type, description || null, id]

);

if (result.affectedRows === 0) {

return res.status(404).json({

success: false,

message: 'Événement non trouvé'

});

}

res.json({

success: true,

message: 'Événement mis à jour avec succès'

});

} catch (error) {

console.error('Erreur modification événement:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// Dans votre fichier server.js

app.get('/api/classes', async (req, res) => {

try {

const { filiere } = req.query;

// Validation du paramètre

if (!filiere || isNaN(filiere)) {

return res.status(400).json({

success: false,

message: "ID de filière invalide"

});

}

// Requête SQL avec jointure pour vérifier l'existence de la filière

const [classes] = await pool.query(`

SELECT c.id, c.nom

FROM classes c

JOIN filieres f ON c.filiere\_id = f.id

WHERE f.id = ?

ORDER BY c.nom

`, [filiere]);

res.json({

success: true,

data: classes

});

} catch (error) {

console.error('Erreur API classes:', error);

res.status(500).json({

success: false,

message: "Erreur serveur"

});

}

});

// Route pour récupérer les documents d'un étudiant

app.get('/api/student-documents', authenticateStudent, async (req, res) => {

try {

const { filiere, classe } = req.query;

// 1. Récupérer les matières de la filière et classe

const [subjects] = await pool.query(`

SELECT m.id, m.nom, s.numero AS semestre

FROM matieres m

JOIN semestres s ON m.semestre\_id = s.id

JOIN classes c ON s.classe\_id = c.id

JOIN filieres f ON c.filiere\_id = f.id

WHERE f.nom = ? AND c.nom = ?

`, [filiere, classe]);

// 2. Récupérer les documents associés

const [documents] = await pool.query(`

SELECT d.id, d.titre, d.description, d.date\_upload AS date,

d.taille AS size, d.type, d.matiere\_id,

m.nom AS matiere, s.numero AS semestre

FROM documents d

JOIN matieres m ON d.matiere\_id = m.id

JOIN semestres s ON m.semestre\_id = s.id

JOIN classes c ON s.classe\_id = c.id

JOIN filieres f ON c.filiere\_id = f.id

WHERE f.nom = ? AND c.nom = ?

`, [filiere, classe]);

res.json({

success: true,

subjects,

documents: documents.map(doc => ({

...doc,

viewed: false // Vous pourriez suivre les vues dans la base

}))

});

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({ success: false, message: 'Erreur serveur' });

}

});

// Middleware d'authentification étudiant

// Configuration du modèle Emploi

const getEmplois = async () => {

try {

const [emplois] = await pool.query(`

SELECT e.\*,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero

FROM emplois\_du\_temps e

LEFT JOIN filieres f ON e.filiere\_id = f.id

LEFT JOIN classes c ON e.classe\_id = c.id

LEFT JOIN semestres s ON e.semestre\_id = s.id

`);

return emplois;

} catch (err) {

console.error('Erreur récupération emplois:', err);

throw err;

}

};

// Route pour récupérer les cours par filière et classe

app.get('/api/cours-etudiant', authenticateStudent, async (req, res) => {

try {

const { filiere, classe } = req.query;

// 1. Récupérer l'ID de la classe

const [classeData] = await pool.query(

'SELECT id FROM classes WHERE nom = ? LIMIT 1',

[classe]

);

if (!classeData.length) {

return res.status(404).json({

success: false,

message: 'Classe non trouvée'

});

}

const classeId = classeData[0].id;

// 2. Récupérer les semestres de cette classe

const [semestres] = await pool.query(

'SELECT id, numero FROM semestres WHERE classe\_id = ?',

[classeId]

);

// 3. Pour chaque semestre, récupérer les matières

const result = await Promise.all(semestres.map(async (semestre) => {

const [matieres] = await pool.query(`

SELECT m.id, m.nom, m.credits,

e.Nom\_et\_prénom AS enseignant

FROM matieres m

LEFT JOIN enseignants e ON m.enseignant\_id = e.CIN

WHERE m.semestre\_id = ?

`, [semestre.id]);

return {

semestre: semestre.numero,

matieres

};

}));

res.json({

success: true,

data: result

});

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// GET /api/filieres

router.get('/filieres', async (req, res) => {

try {

const [filieres] = await connection.query('SELECT \* FROM filieres');

res.json(filieres);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

// GET /api/classes

router.get('/classes', async (req, res) => {$

try {

const [classes] = await connection.query('SELECT \* FROM classes');

res.json(classes);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

// GET /api/semestres

router.get('/semestres', async (req, res) => {

try {

const [semestres] = await connection.query('SELECT \* FROM semestres');

res.json(semestres);

} catch (err) {

res.status(500).json({ message: err.message });

}

});

router.get('/filieres/:filiereId/classes', async (req, res) => {

try {

const classes = await db.query(

'SELECT \* FROM classes WHERE filiere\_id = $1',

[req.params.filiereId]

);

res.json(classes.rows);

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Exemple correct dans une route

app.get('/api/test-filieres', async (req, res) => {

try {

const response = await axios.get('http://localhost:3001/api/filieres');

res.json(response.data);

} catch (error) {

console.error('Erreur Axios:', error);

res.status(500).send('Erreur serveur');

}

});

app.get("/api/emplois", async (req, res) => {

try {

const [emplois] = await pool.query(`

SELECT e.\*,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero

FROM emplois\_du\_temps e

LEFT JOIN filieres f ON e.filiere\_id = f.id

LEFT JOIN classes c ON e.classe\_id = c.id

LEFT JOIN semestres s ON e.semestre\_id = s.id

ORDER BY e.created\_at DESC

`);

console.log("Emplois récupérés:", emplois); // Log pour débogage

res.json({

success: true,

data: emplois // Assurez-vous que c'est bien 'data' et non 'emplois'

});

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({

success: false,

message: "Erreur serveur"

});

}

});

// Route pour récupérer les filières, classes et matières

app.get('/api/teaching-data', async (req, res) => {

try {

// 1. Récupération des données avec gestion des erreurs de requête

const [filieres] = await pool.query('SELECT id, nom FROM filieres ORDER BY nom');

const [classes] = await pool.query(`

SELECT

c.id,

c.nom,

c.filiere\_id,

f.nom AS filiere\_nom

FROM classes c

INNER JOIN filieres f ON c.filiere\_id = f.id

ORDER BY c.nom

`);

const [matieres] = await pool.query(`

SELECT

m.id,

m.nom,

s.id AS semestre\_id,

s.numero AS semestre\_numero,

c.id AS classe\_id,

c.nom AS classe\_nom,

f.id AS filiere\_id,

f.nom AS filiere\_nom

FROM matieres m

INNER JOIN semestres s ON m.semestre\_id = s.id

INNER JOIN classes c ON s.classe\_id = c.id

INNER JOIN filieres f ON c.filiere\_id = f.id

ORDER BY m.nom

`);

// 2. Structuration de la réponse

const response = {

success: true,

data: {

filieres: filieres.map(f => ({

id: f.id,

nom: f.nom

})),

classes: classes.map(c => ({

id: c.id,

nom: c.nom,

filiere\_id: c.filiere\_id,

filiere\_nom: c.filiere\_nom

})),

matieres: matieres.map(m => ({

id: m.id,

nom: m.nom,

semestre: {

id: m.semestre\_id,

numero: m.semestre\_numero

},

classe: {

id: m.classe\_id,

nom: m.classe\_nom

},

filiere: {

id: m.filiere\_id,

nom: m.filiere\_nom

}

}))

}

};

// 3. Journalisation contrôlée

if (process.env.NODE\_ENV === 'development') {

console.log('Données pédagogiques chargées:', {

filieres: response.data.filieres,

classes: response.data.classes,

matieres: response.data.matieres

});

}

res.json(response);

} catch (error) {

// 4. Gestion d'erreur améliorée

console.error(`Erreur teaching-data: ${error.message}`);

res.status(500).json({

success: false,

message: 'Erreur de chargement des données pédagogiques',

error: process.env.NODE\_ENV === 'production' ? undefined : error.message

});

}

});

app.get('/api/filieres', async (req, res) => {

try {

const [filieres] = await pool.query('SELECT id, nom FROM filieres');

res.json(filieres);

} catch (error) {

res.status(500).json({ error: error.message });

}

});

// Et pour les classes :

app.get('/api/classes', async (req, res) => {

try {

const [classes] = await pool.query(`

SELECT c.id, c.nom, f.nom as filiere\_nom, f.id as filiere\_id

FROM classes c

JOIN filieres f ON c.filiere\_id = f.id

`);

res.json({

success: true,

data: classes

});

} catch (err) {

console.error(err);

res.status(500).json({

success: false,

error: 'Erreur serveur'

});

}

});

// Corrigez la route pour retourner le bon format de données

app.get('/api/enseignants/list', async (req, res) => {

try {

const [enseignants] = await pool.query('SELECT CIN, Nom\_et\_prénom FROM enseignants');

res.json({

success: true,

data: enseignants // Assurez-vous de retourner un objet avec propriété data

});

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur',

error: process.env.NODE\_ENV === 'development' ? error.message : undefined

});

}

});

app.get("/api/emplois/enseignant/:cin", authenticate(['enseignant']), async (req, res) => {

try {

const { cin } = req.params;

const [emplois] = await pool.query(`

SELECT

e.id,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero,

e.fichier\_path,

e.published\_at

FROM emplois\_du\_temps e

LEFT JOIN filieres f ON e.filiere\_id = f.id

LEFT JOIN classes c ON e.classe\_id = c.id

LEFT JOIN semestres s ON e.semestre\_id = s.id

WHERE e.enseignant\_id = ? AND e.type = 'enseignant' AND e.published = TRUE

ORDER BY e.published\_at DESC

`, [cin]);

res.json({

success: true,

data: emplois

});

} catch (error) {

console.error("Erreur récupération emploi enseignant :", error);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

app.get('/api/emplois/enseignant/:cin/download', async (req, res) => {

try {

const { cin } = req.params;

const [emploi] = await pool.query(

'SELECT fichier\_path FROM emplois\_du\_temps WHERE enseignant\_id = ? AND published = 1 ORDER BY created\_at DESC LIMIT 1',

[cin]

);

if (!emploi.length) {

return res.status(404).json({ success: false, message: "Emploi non trouvé" });

}

const filePath = path.join(\_\_dirname, emploi[0].fichier\_path);

res.download(filePath);

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({ success: false, message: 'Erreur serveur' });

}

});

// Route pour l'emploi du temps étudiant

app.get("/api/etudiant/emplois", async (req, res) => {

try {

const { filiere, classe } = req.query;

const [emplois] = await pool.query(`

SELECT e.\*,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero

FROM emplois\_du\_temps e

LEFT JOIN filieres f ON e.filiere\_id = f.id

LEFT JOIN classes c ON e.classe\_id = c.id

LEFT JOIN semestres s ON e.semestre\_id = s.id

WHERE e.type = 'etudiant'

AND f.nom = ?

AND c.nom = ?

AND e.published = 1

ORDER BY e.created\_at DESC

`, [filiere, classe]);

res.json({

success: true,

data: emplois

});

} catch (error) {

console.error("Erreur:", error);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

// Dans votre fichier backend (ex: server.js)

app.get('/api/emplois/:id/parsed', async (req, res) => {

try {

const { id } = req.params;

const [rows] = await pool.query(

'SELECT fichier\_path FROM emplois\_du\_temps WHERE id = ?',

[id]

);

if (!rows.length || !rows[0].fichier\_path) {

return res.status(404).json({ success: false, message: "Fichier introuvable" });

}

const filePath = path.join(\_\_dirname, rows[0].fichier\_path);

if (!fs.existsSync(filePath)) {

return res.status(404).json({ success: false, message: "Fichier manquant sur le serveur" });

}

const workbook = XLSX.readFile(filePath);

const sheetName = workbook.SheetNames[0];

const data = XLSX.utils.sheet\_to\_json(workbook.Sheets[sheetName], { defval: '' });

res.json({ success: true, data });

} catch (error) {

console.error("Erreur parsing emploi:", error);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

router.get('/auth/verify', authenticateAdmin, (req, res) => {

res.json({

valid: true,

user: {

email: req.user.email,

role: req.user.role

}

});

});

app.get('/api/enseignants', async (req, res) => {

try {

const [enseignants] = await pool.query(

'SELECT CIN, Nom\_et\_prénom FROM enseignants ORDER BY Nom\_et\_prénom'

);

res.json({

success: true,

data: enseignants

});

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

app.post("/auth/login", async (req, res) => {

const { email, password } = req.body;

try {

// Admin login

const [admin] = await pool.query("SELECT \* FROM admin WHERE Email = ?", [email]);

if (admin.length && password === admin[0].password) {

const token = jwt.sign({ email, role: 'admin' }, JWT\_SECRET, { expiresIn: '24h' });

return res.json({

success: true,

token,

user: { email, role: 'admin' }

});

}

// Agent login

const [agent] = await pool.query("SELECT \* FROM agents WHERE email = ?", [email]);

if (agent.length && password === agent[0].password) {

const user = agent[0];

const token = jwt.sign({ id: user.id, role: user.role }, JWT\_SECRET, { expiresIn: '24h' });

return res.json({

success: true,

token,

user: {

id: user.id,

nom: user.nom,

prenom: user.prenom,

email: user.email,

role: user.role,

departement: user.departement

}

});

}

return res.status(401).json({ success: false, message: "Identifiants incorrects" });

} catch (err) {

console.error("Erreur login unifié :", err);

return res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

// Routes pour la gestion des examens par l'admin

// Route pour créer un examen (admin seulement)

app.post('/api/examens', async (req, res) => {

try {

const {

matiere\_id,

filiere\_id,

classe\_id,

semestre\_id,

date,

heure\_debut,

heure\_fin,

salle,

enseignant\_id,

type

} = req.body;

// Validation des données

if (!matiere\_id || !filiere\_id || !classe\_id || !semestre\_id || !date ||

!heure\_debut || !heure\_fin || !salle || !type) {

return res.status(400).json({

success: false,

message: 'Tous les champs obligatoires doivent être remplis'

});

}

// Vérifier que la date est dans le futur

const examDate = new Date(`${date}T${heure\_debut}`);

if (examDate < new Date()) {

return res.status(400).json({

success: false,

message: 'La date et heure de l\'examen doivent être dans le futur'

});

}

// Insertion dans la base de données

const [result] = await pool.query(

`INSERT INTO examens

(matiere\_id, filiere\_id, classe\_id, semestre\_id, date, heure\_debut, heure\_fin, salle, enseignant\_id, type)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,

[

matiere\_id,

filiere\_id,

classe\_id,

semestre\_id,

date,

heure\_debut,

heure\_fin,

salle,

enseignant\_id || null,

type

]

);

res.status(201).json({

success: true,

message: 'Examen créé avec succès',

examen\_id: result.insertId

});

} catch (error) {

console.error('Erreur création examen:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur lors de la création de l\'examen',

error: process.env.NODE\_ENV === 'development' ? error.message : undefined

});

}

});

// Route pour lister tous les examens (admin seulement)

app.get('/api/examens', async (req, res) => {

console.log("Reçu requête GET /api/examens"); // Log 1

try {

const [rows] = await pool.query(`

SELECT e.\*, m.nom AS matiere\_nom

FROM examens e

LEFT JOIN matieres m ON e.matiere\_id = m.id

`);

console.log("Données récupérées:", rows); // Log 2

res.json({

success: true,

data: rows,

message: `${rows.length} examens trouvés`

});

} catch (error) {

console.error("Erreur complète:", error); // Log 3

res.status(500).json({

success: false,

message: "Échec de la récupération",

error: error.message

});

}

});

// Route pour publier/diffuser un examen (admin seulement)

app.put('/api/examens/:id/publish', async (req, res) => {

try {

const { id } = req.params;

const { cible } = req.body;

if (!['enseignants', 'etudiants', 'tous'].includes(cible)) {

return res.status(400).json({

success: false,

message: 'Cible de diffusion invalide'

});

}

// Mettre à jour les flags de diffusion

const updates = {

diffusion\_at: new Date()

};

if (cible === 'enseignants' || cible === 'tous') {

updates.diffusion\_enseignants = true;

}

if (cible === 'etudiants' || cible === 'tous') {

updates.diffusion\_etudiants = true;

}

const [result] = await pool.query(

`UPDATE examens SET

diffusion\_enseignants = ?,

diffusion\_etudiants = ?,

diffusion\_at = ?

WHERE id = ?`,

[

updates.diffusion\_enseignants || false,

updates.diffusion\_etudiants || false,

updates.diffusion\_at,

id

]

);

if (result.affectedRows === 0) {

return res.status(404).json({

success: false,

message: 'Examen non trouvé'

});

}

res.json({

success: true,

message: `Examen diffusé avec succès aux ${cible}`

});

} catch (error) {

console.error('Erreur publication examen:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur lors de la publication de l\'examen'

});

}

});

// Route pour modifier un examen (admin seulement)

app.put('/api/examens/:id', authenticateAdmin, async (req, res) => {

try {

const { id } = req.params;

const {

matiere\_id,

filiere\_id,

classe\_id,

semestre\_id,

date,

heure\_debut,

heure\_fin,

salle,

enseignant\_id,

type

} = req.body;

// Vérifier que l'examen existe

const [examen] = await pool.query('SELECT id FROM examens WHERE id = ?', [id]);

if (!examen.length) {

return res.status(404).json({

success: false,

message: 'Examen non trouvé'

});

}

// Mise à jour dans la base de données

const [result] = await pool.query(

`UPDATE examens SET

matiere\_id = ?,

filiere\_id = ?,

classe\_id = ?,

semestre\_id = ?,

date = ?,

heure\_debut = ?,

heure\_fin = ?,

salle = ?,

enseignant\_id = ?,

type = ?

WHERE id = ?`,

[

matiere\_id,

filiere\_id,

classe\_id,

semestre\_id,

date,

heure\_debut,

heure\_fin,

salle,

enseignant\_id || null,

type,

id

]

);

res.json({

success: true,

message: 'Examen mis à jour avec succès'

});

} catch (error) {

console.error('Erreur modification examen:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur lors de la modification de l\'examen'

});

}

});

// Route pour supprimer un examen (admin seulement)

app.delete('/api/examens/:id', authenticateAdmin, async (req, res) => {

try {

const { id } = req.params;

// Vérifier que l'examen existe

const [examen] = await pool.query('SELECT id FROM examens WHERE id = ?', [id]);

if (!examen.length) {

return res.status(404).json({

success: false,

message: 'Examen non trouvé'

});

}

// Suppression de la base de données

await pool.query('DELETE FROM examens WHERE id = ?', [id]);

res.json({

success: true,

message: 'Examen supprimé avec succès'

});

} catch (error) {

console.error('Erreur suppression examen:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur lors de la suppression de l\'examen'

});

}

});

// Route pour récupérer les examens diffusés aux enseignants

app.get('/api/examens/enseignants', authenticate(['enseignant']), async (req, res) => {

try {

const [examens] = await pool.query(`

SELECT e.\*,

m.nom AS matiere\_nom,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero

FROM examens e

LEFT JOIN matieres m ON e.matiere\_id = m.id

LEFT JOIN filieres f ON e.filiere\_id = f.id

LEFT JOIN classes c ON e.classe\_id = c.id

LEFT JOIN semestres s ON e.semestre\_id = s.id

WHERE e.diffusion\_enseignants = TRUE

AND e.date >= CURDATE()

ORDER BY e.date ASC, e.heure\_debut ASC

`);

res.json({

success: true,

data: examens

});

} catch (error) {

console.error('Erreur récupération examens enseignants:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur lors de la récupération des examens'

});

}

});

// Route pour récupérer les examens diffusés aux étudiants

app.get('/api/examens/etudiants', authenticate(['etudiant']), async (req, res) => {

try {

// Récupérer la filière et classe de l'étudiant

const [etudiant] = await pool.query(

'SELECT Filière AS filiere\_id, Classe AS classe\_id FROM etudiant WHERE CIN = ?',

[req.user.cin]

);

if (!etudiant.length) {

return res.status(404).json({

success: false,

message: 'Étudiant non trouvé'

});

}

const { filiere\_id, classe\_id } = etudiant[0];

const [examens] = await pool.query(`

SELECT e.\*,

m.nom AS matiere\_nom,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero,

en.Nom\_et\_prénom AS enseignant\_nom

FROM examens e

LEFT JOIN matieres m ON e.matiere\_id = m.id

LEFT JOIN filieres f ON e.filiere\_id = f.id

LEFT JOIN classes c ON e.classe\_id = c.id

LEFT JOIN semestres s ON e.semestre\_id = s.id

LEFT JOIN enseignants en ON e.enseignant\_id = en.CIN

WHERE e.diffusion\_etudiants = TRUE

AND e.filiere\_id = ?

AND e.classe\_id = ?

AND e.date >= CURDATE()

ORDER BY e.date ASC, e.heure\_debut ASC

`, [filiere\_id, classe\_id]);

res.json({

success: true,

data: examens

});

} catch (error) {

console.error('Erreur récupération examens étudiants:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur lors de la récupération des examens'

});

}

});

app.get("/api/examens/etudiant/:cin", async (req, res) => {

const { cin } = req.params;

try {

const [examens] = await pool.query(`

SELECT

m.nom AS matiere\_nom,

e.date,

e.heure\_debut,

e.heure\_fin,

e.salle,

e.type

FROM examens e

JOIN matieres m ON e.matiere\_id = m.id

JOIN etudiant et ON et.CIN = ?

JOIN classes c ON c.nom = et.Classe

WHERE e.filiere\_id = c.filiere\_id

AND e.classe\_id = c.id

AND e.diffusion\_etudiants = 1

`, [cin]);

res.status(200).json({ success: true, data: examens });

} catch (error) {

console.error('Erreur récupération examens étudiant :', error);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});

// Route pour récupérer les examens d'un enseignant spécifique

app.get('/api/examens/enseignant/:cin', authenticate(['enseignant']), async (req, res) => {

try {

const { cin } = req.params;

const [examens] = await pool.query(`

SELECT e.\*,

m.nom AS matiere\_nom,

f.nom AS filiere\_nom,

c.nom AS classe\_nom,

s.numero AS semestre\_numero

FROM examens e

JOIN matieres m ON e.matiere\_id = m.id

JOIN filieres f ON e.filiere\_id = f.id

JOIN classes c ON e.classe\_id = c.id

JOIN semestres s ON e.semestre\_id = s.id

WHERE e.enseignant\_id = ?

AND e.diffusion\_enseignants = TRUE

AND e.date >= CURDATE()

ORDER BY e.date ASC, e.heure\_debut ASC

`, [cin]);

res.json({

success: true,

data: examens

});

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({

success: false,

message: 'Erreur serveur'

});

}

});

// ✅ Route pour récupérer les examens d'un étudiant

{/\*app.get("/api/examens/etudiant/:cin", async (req, res) => {

const { cin } = req.params;

try {

const [examens] = await pool.query(`

SELECT

m.nom AS matiere\_nom,

e.date\_examen AS date,

e.heure\_debut,

e.heure\_fin,

e.salle,

e.type

FROM examens e

JOIN matieres m ON e.matiere\_id = m.id

JOIN etudiant et ON et.CIN = ?

JOIN classes c ON c.nom = et.Classe

WHERE e.filiere\_id = c.filiere\_id

AND e.classe\_id = c.id

AND e.diffusion\_etudiants = 1

`, [cin]);

res.status(200).json({ success: true, data: examens });

} catch (error) {

console.error('Erreur récupération examens étudiant :', error);

res.status(500).json({ success: false, message: "Erreur serveur" });

}

});\*/}

//Ageeeeeeeents

// GET all agents

app.get('/agents', async (req, res) => {

try {

const [rows] = await pool.query('SELECT \* FROM agents');

res.json(rows);

} catch (err) {

res.status(500).json({ error: 'Erreur lors de la récupération des agents' });

}

});

// POST create agent

app.post('/agents', async (req, res) => {

const { nom, prenom, email, password, departement, role } = req.body;

if (!nom || !prenom || !email || !password || !departement || !role) {

return res.status(400).json({ error: 'Champs obligatoires manquants' });

}

try {

const hashedPassword = await bcrypt.hash(password, 10);

const [result] = await pool.query(

'INSERT INTO agents (nom, prenom, email, password, departement, role) VALUES (?, ?, ?, ?, ?, ?)',

[nom, prenom, email, hashedPassword, departement, role]

);

res.status(201).json({ id: result.insertId, message: 'Agent ajouté avec succès' });

} catch (err) {

res.status(500).json({ error: 'Erreur lors de l\'ajout de l\'agent' });

}

});

// DELETE agent

app.delete('/agents/:id', async (req, res) => {

const { id } = req.params;

try {

await pool.query('DELETE FROM agents WHERE id = ?', [id]);

res.json({ message: 'Agent supprimé avec succès' });

} catch (err) {

res.status(500).json({ error: 'Erreur lors de la suppression' });

}

});

// PUT update agent

app.put('/agents/:id', async (req, res) => {

const { id } = req.params;

const { nom, prenom, email, password, departement, role } = req.body;

if (!nom || !prenom || !email || !departement || !role) {

return res.status(400).json({ error: 'Champs requis manquants' });

}

try {

let query, params;

if (password && password.trim() !== '') {

// Si un nouveau mot de passe est fourni

const hashedPassword = await bcrypt.hash(password, 10);

query = `

UPDATE agents

SET nom = ?, prenom = ?, email = ?, password = ?, departement = ?, role = ?

WHERE id = ?

`;

params = [nom, prenom, email, hashedPassword, departement, role, id];

} else {

// Sinon ne pas modifier le mot de passe

query = `

UPDATE agents

SET nom = ?, prenom = ?, email = ?, departement = ?, role = ?

WHERE id = ?

`;

params = [nom, prenom, email, departement, role, id];

}

const [result] = await pool.query(query, params);

if (result.affectedRows === 0) {

return res.status(404).json({ error: 'Agent non trouvé' });

}

res.json({ message: 'Agent mis à jour avec succès' });

} catch (err) {

console.error(err);

res.status(500).json({ error: 'Erreur lors de la mise à jour de l\'agent' });

}

});

app.post('/api/agents/login', async (req, res) => {

const { email, password } = req.body;

if (!email || !password) {

return res.status(400).json({ success: false, message: 'Email et mot de passe requis' });

}

try {

const [rows] = await pool.query('SELECT \* FROM agents WHERE email = ?', [email]);

const agent = rows[0];

if (!agent) {

return res.status(404).json({ success: false, message: 'Agent introuvable' });

}

const passwordMatch = await bcrypt.compare(password, agent.password);

if (!passwordMatch) {

return res.status(401).json({ success: false, message: 'Mot de passe incorrect' });

}

const token = jwt.sign(

{

id: agent.id,

email: agent.email,

role: agent.role

},

JWT\_SECRET,

{ expiresIn: '7d' }

);

res.json({

success: true,

message: 'Connexion réussie',

token,

agent: {

id: agent.id,

nom: agent.nom,

prenom: agent.prenom,

email: agent.email,

role: agent.role,

departement: agent.departement

}

});

} catch (err) {

console.error(err);

res.status(500).json({ success: false, message: 'Erreur serveur' });

}

});

app.get('/api/agents/profile', authenticateAgent(), async (req, res) => {

try {

const [results] = await pool.query(

'SELECT id, nom, prenom, email, departement, role FROM agents WHERE id = ?',

[req.user.id]

);

if (results.length === 0) {

return res.status(404).json({ success: false, message: 'Agent non trouvé' });

}

res.json({ success: true, data: results[0] });

} catch (error) {

console.error('Erreur:', error);

res.status(500).json({ success: false, message: 'Erreur serveur' });

}

});

// Protégez vos routes

app.get("/api/protected-route", authenticate(['enseignant']), async (req, res) => {

res.json({ message: "Accès autorisé" });

});

// Démarrer le serveur

const PORT = 5000;

app.listen(PORT, () => {

console.log(`🚀 Serveur en écoute sur http://localhost:${PORT}`);

}).on('error', (err) => {

console.error('❌ Erreur du serveur:', err.message);

});