'use strict';

const express = require('express');

const router = express.Router();

const { MiseAJour, User, Cadre, Absence, HistoriqueStatsJournalieresCadres, Eleve } = require('../models'); // Assurez-vous d'ajouter Eleve si vous l'avez

// Assurez-vous que authenticateJWT et isAdmin sont correctement importés de votre middleware

const { authenticateJWT, isAdmin } = require('../middleware/authMiddleware');

const { Op } = require('sequelize');

// Importez les fonctions de date qui gèrent la "journée historique"

const { getHistoricalDate, getHistoricalDayStartTime, getHistoricalDayEndTime } = require('../utils/date');

const miseAJourController = require('../controllers/miseAJourController'); // Assurez-vous que ce fichier existe

const Sequelize = require('sequelize'); // Importez Sequelize pour utiliser Sequelize.col, Sequelize.fn, etc.

// NOUVEAU : Importez la fonction de calcul des statistiques depuis historicalTasks.js

const { calculateCurrentAggregateStats } = require('../tasks/historicalTasks'); // <-- VÉRIFIEZ CE CHEMIN !

// Middleware d'authentification appliqué à toutes les routes de ce routeur

router.use(authenticateJWT);

// --- ROUTES POUR LA GESTION DES MISES À JOUR / VALIDATION ---

// POST /api/mises-a-jour/submit

router.post('/submit', async (*req*, *res*) => {

    console.log('[MiseAJourRoutes] POST /submit - Début de la soumission de la mise à jour.');

    const { update\_date } = *req*.body;

    if (!update\_date) {

        console.warn('[MiseAJourRoutes] POST /submit - Erreur: update\_date est manquant.');

        return *res*.status(400).json({ message: 'La date de la mise à jour (update\_date) est requise.' });

    }

    try {

        console.log(`[MiseAJourRoutes] POST /submit - Recherche de l'utilisateur avec cadre pour User ID: ${*req*.user.id}`);

        const userWithCadre = await User.findByPk(*req*.user.id, {

            // CORRECTION: Utilisez l'alias 'cadre' (minuscule 'c') tel que défini dans models/user.js

            include: [{ model: Cadre, as: 'cadre', attributes: ['id', 'service', 'responsibility\_scope', 'responsible\_escadron\_id'] }]

        });

        if (!userWithCadre || !userWithCadre.cadre) { // CORRECTION: Accédez avec l'alias en minuscule

            console.warn(`[MiseAJourRoutes] POST /submit - Utilisateur ${*req*.user.username} (ID: ${*req*.user.id}) a tenté de soumettre une mise à jour sans être lié à un cadre.`);

            return *res*.status(403).json({ message: 'Vous devez être lié à un cadre pour soumettre une mise à jour.' });

        }

        const cadreIdSoumetteur = userWithCadre.cadre.id; // CORRECTION: Accédez avec l'alias en minuscule

        console.log(`[MiseAJourRoutes] POST /submit - Cadre Soumetteur ID: ${cadreIdSoumetteur}, Date de mise à jour: ${update\_date}`);

        const nouvelleSoumission = await MiseAJour.create({

            submitted\_by\_id: *req*.user.id,

            update\_date: update\_date,

            cadre\_id: cadreIdSoumetteur,

            status: 'Validée',

            validated\_by\_id: *req*.user.id,

            validation\_date: **new** *Date*()

        });

        console.log(`[MiseAJourRoutes] POST /submit - Nouvelle soumission créée avec succès. ID: ${nouvelleSoumission.id}, Statut: ${nouvelleSoumission.status}`);

        const soumissionAvecDetails = await MiseAJour.findByPk(nouvelleSoumission.id, {

            include: [

                { model: User, as: 'SubmittedBy', attributes: ['id', 'username', 'nom', 'prenom'] },

                // L'alias 'Cadre' ici est correct si c'est ainsi que MiseAJour est associé à Cadre dans son propre modèle.

                { model: Cadre, as: 'Cadre', attributes: ['id', 'service', 'fonction', 'responsibility\_scope', 'responsible\_escadron\_id'] }

            ]

        });

        console.log(`[MiseAJourRoutes] POST /submit - Détails de la soumission récupérés pour ID: ${soumissionAvecDetails.id}`);

        if (nouvelleSoumission.status === 'Validée') {

            console.log(`[MiseAJourRoutes] POST /submit - La soumission est "Validée". Déclenchement de la mise à jour des statuts des personnes.`);

            const cadreSoumetteur = soumissionAvecDetails.Cadre; // Accédez avec l'alias défini dans MiseAJour (supposé 'Cadre' majuscule ici)

            const dateMiseAJour = nouvelleSoumission.update\_date;

            if (cadreSoumetteur) {

                const scopeType = cadreSoumetteur.responsibility\_scope;

                let cadresToUpdate = [];

                let elevesToUpdate = [];

                console.log(`[MiseAJourRoutes] POST /submit - Scope de responsabilité du cadre: ${scopeType}`);

                if (scopeType === 'Service' && cadreSoumetteur.service) {

                    console.log(`[MiseAJourRoutes] POST /submit - Recherche de cadres pour le service: ${cadreSoumetteur.service}`);

                    cadresToUpdate = await Cadre.findAll({

                        where: { service: cadreSoumetteur.service },

                        attributes: ['id']

                    });

                    console.log(`[MiseAJourRoutes] POST /submit - Trouvé ${cadresToUpdate.length} cadres dans le service.`);

                } else if (scopeType === 'Escadron' && cadreSoumetteur.responsible\_escadron\_id) {

                    console.log(`[MiseAJourRoutes] POST /submit - Recherche de cadres pour l'escadron: ${cadreSoumetteur.responsible\_escadron\_id}`);

                    cadresToUpdate = await Cadre.findAll({

                        where: { responsible\_escadron\_id: cadreSoumetteur.responsible\_escadron\_id },

                        attributes: ['id']

                    });

                    console.log(`[MiseAJourRoutes] POST /submit - Trouvé ${cadresToUpdate.length} cadres dans l'escadron.`);

                    // SI VOUS AVEZ UNE TABLE ELEVE SÉPARÉE, DÉCOMMENTEZ ET AJUSTEZ CELA :

                    // console.log(`[MiseAJourRoutes] POST /submit - Recherche d'élèves pour l'escadron: ${cadreSoumetteur.responsible\_escadron\_id}`);

                    // elevesToUpdate = await Eleve.findAll({

                    //     where: { escadron\_id: cadreSoumetteur.responsible\_escadron\_id },

                    //     attributes: ['id']

                    // });

                    // console.log(`[MiseAJourRoutes] POST /submit - Trouvé ${elevesToUpdate.length} élèves dans l'escadron.`);

                } else {

                    console.warn(`[MiseAJourRoutes] POST /submit - Cadre soumetteur (ID: ${cadreSoumetteur.id}) a un scope de responsabilité "${scopeType}" invalide ou incomplet pour la soumission ${nouvelleSoumission.id}. Skipping status update.`);

                }

                if (cadresToUpdate.length > 0) {

                    const cadreIds = cadresToUpdate.map(*c* => c.id);

                    console.log(`[MiseAJourRoutes] POST /submit - Mise à jour des statuts pour les cadres IDs: ${cadreIds.join(', ')}`);

                    await Cadre.update(

                        {

                            statut\_absence: 'Présent',

                            date\_debut\_absence: null,

                            motif\_absence: null,

                            motif\_details: null,

                            timestamp\_derniere\_maj\_statut: **new** Date()

                        },

                        {

                            where: { id: { [Op.in]: cadreIds } }

                        }

                    );

                    console.log(`[MiseAJourRoutes] POST /submit - Statuts de ${cadresToUpdate.length} cadres mis à jour à 'Présent' pour l'entité ${scopeType} (${cadreSoumetteur.service || cadreSoumetteur.responsible\_escadron\_id}) à la date ${dateMiseAJour}.`);

                }

            }

        }

        console.log('[MiseAJourRoutes] POST /submit - Fin de la soumission de la mise à jour avec succès.');

        return res.status(201).json({ message: 'Mise à jour soumise avec succès', soumission: soumissionAvecDetails });

    } catch (error) {

        console.error('[MiseAJourRoutes] POST /submit - Erreur lors de la soumission de la mise à jour :', error);

        if (error.name === 'SequelizeValidationError') {

            return res.status(400).json({ message: 'Erreur de validation : ' + error.errors.map(*e* => e.message).join(', '), errors: error.errors });

        }

        return res.status(500).json({ message: 'Erreur serveur lors de la soumission de la mise à jour.' });

    }

});

// GET /api/mises-a-jour/status-for-cadre

router.get('/status-for-cadre', async (*req*, *res*) => {

    console.log('[MiseAJourRoutes] GET /status-for-cadre - Vérification du statut de soumission.');

    const { date, cadreId } = req.query;

    if (!date) {

        console.warn('[MiseAJourRoutes] GET /status-for-cadre - Erreur: Le paramètre "date" est manquant.');

        return res.status(400).json({ message: 'Le paramètre "date" est requis.' });

    }

    try {

        let targetCadreId = cadreId;

        console.log(`[MiseAJourRoutes] GET /status-for-cadre - Date demandée: ${date}, Cadre ID fourni: ${cadreId || 'N/A'}`);

        if (req.user.role !== 'Admin') {

            console.log(`[MiseAJourRoutes] GET /status-for-cadre - Utilisateur non-Admin (Rôle: ${req.user.role}). Récupération du cadre de l'utilisateur.`);

            const userWithCadre = await User.findByPk(req.user.id, {

                // CORRECTION: Utilisez l'alias 'cadre' (minuscule 'c')

                include: [{ model: Cadre, as: 'cadre', attributes: ['id'] }]

            });

            if (!userWithCadre || !userWithCadre.cadre) { // CORRECTION: Accédez avec l'alias en minuscule

                console.warn(`[MiseAJourRoutes] GET /status-for-cadre - Utilisateur ${req.user.username} (ID: ${req.user.id}) a tenté de vérifier le statut sans être lié à un cadre.`);

                return res.status(403).json({ message: 'Vous devez être lié à un cadre pour vérifier les statuts.' });

            }

            targetCadreId = userWithCadre.cadre.id; // CORRECTION: Accédez avec l'alias en minuscule

            console.log(`[MiseAJourRoutes] GET /status-for-cadre - Cadre de l'utilisateur (ID: ${req.user.id}): ${targetCadreId}`);

            if (cadreId && parseInt(cadreId, 10) !== targetCadreId) {

                console.warn(`[MiseAJourRoutes] GET /status-for-cadre - Utilisateur Standard ${req.user.username} (ID: ${req.user.id}) a tenté de vérifier le statut pour un cadre différent (ID: ${cadreId}).`);

                return res.status(403).json({ message: 'Vous n\'êtes pas autorisé à vérifier le statut de soumission pour ce cadre.' });

            }

        } else {

            if (!targetCadreId) {

                console.warn('[MiseAJourRoutes] GET /status-for-cadre - Erreur: Pour un administrateur, cadreId est requis si non lié à un cadre spécifique.');

                return res.status(400).json({ message: 'Pour un administrateur, le paramètre "cadreId" est requis.' });

            }

        }

        const soumissionExistante = await MiseAJour.findOne({

            where: {

                cadre\_id: targetCadreId,

                update\_date: date,

                status: 'Validée' // On ne considère que les soumissions validées

            }

        });

        const isSubmitted = !!soumissionExistante;

        console.log(`[MiseAJourRoutes] GET /status-for-cadre - Statut de soumission pour cadre ${targetCadreId} à la date ${date}: ${isSubmitted ? 'Soumis' : 'Non Soumis'}`);

        return res.status(200).json({ submitted: isSubmitted });

    } catch (error) {

        console.error('[MiseAJourRoutes] GET /status-for-cadre - Erreur lors de la vérification du statut de soumission :', error);

        return res.status(500).json({ message: 'Erreur serveur lors de la vérification du statut de soumission.' });

    }

});

// GET /api/mises-a-jour/daily-updates

router.get('/daily-updates', async (*req*, *res*) => {

    console.log('[MiseAJourRoutes] GET /daily-updates - Récupération des mises à jour quotidiennes.');

    const { date } = req.query; // La date est optionnelle, par défaut aujourd'hui

    const today = **new** Date();

    const targetDate = date ? **new** Date(date) : today;

    // Utilisation de vos utilitaires de date pour la journée historique

    const startOfDayUTC = getHistoricalDayStartTime(targetDate);

    const endOfDayUTC = getHistoricalDayEndTime(targetDate);

    console.log(`[MiseAJourRoutes] GET /daily-updates - Période de recherche (UTC) : ${startOfDayUTC.toISOString()} à ${endOfDayUTC.toISOString()}`);

    try {

        const completedSubmissions = await MiseAJour.findAll({

            attributes: [

                [Sequelize.col('SubmittedBy.id'), 'id'],

                [Sequelize.col('SubmittedBy.username'), 'username'],

                [Sequelize.fn('COUNT', Sequelize.literal('\*')), 'dailySubmissionCount'],

                [Sequelize.fn('MAX', Sequelize.col('MiseAJour.created\_at')), 'updatedAt']

            ],

            where: {

                created\_at: {

                    [Op.between]: [startOfDayUTC, endOfDayUTC]

                },

                status: 'Validée'

            },

            include: [{

                model: User,

                as: 'SubmittedBy',

                attributes: [], // Pas besoin d'attributs ici, ils sont dans les `attributes` de la requête principale

                where: {

                    status: 'Active',

                    role: 'Standard'

                }

            }],

            group: ['SubmittedBy.id', 'SubmittedBy.username'],

            order: [[Sequelize.fn('MAX', Sequelize.col('MiseAJour.created\_at')), 'DESC']]

        });

        const completedCadreIds = completedSubmissions.map(*s* => s.get('id'));

        console.log(`[MiseAJourRoutes] GET /daily-updates - Cadres "Terminés" (IDs): ${completedCadreIds.join(', ')}`);

        // Récupérer tous les cadres "Standard" actifs

        const allStandardUsers = await User.findAll({

            where: {

                status: 'Active',

                role: 'Standard'

            },

            attributes: ['id', 'username']

        });

        const allStandardUserIds = allStandardUsers.map(*u* => u.id);

        const pendingCadreIds = allStandardUserIds.filter(*id* => !completedCadreIds.includes(id));

        console.log(`[MiseAJourRoutes] GET /daily-updates - Cadres "En attente" (IDs): ${pendingCadreIds.join(', ')}`);

        // Obtenir les détails complets pour les cadres en attente

        const pendingCadreDetails = await User.findAll({

            where: {

                id: { [Op.in]: pendingCadreIds }

            },

            attributes: ['id', 'username', 'nom', 'prenom'],

            // CORRECTION: Utilisez l'alias 'cadre' (minuscule 'c')

            include: [{

                model: Cadre,

                as: 'cadre', // Alias en minuscule

                attributes: ['id', 'service', 'fonction', 'responsibility\_scope', 'responsible\_escadron\_id']

            }]

        });

        // Formater la réponse pour inclure les cadres terminés et en attente

        const response = {

            completed: completedSubmissions.map(*s* => ({

                id: *s*.get('id'),

                username: *s*.get('username'),

                dailySubmissionCount: *s*.get('dailySubmissionCount'),

                updatedAt: *s*.get('updatedAt')

            })),

            pending: pendingCadreDetails.map(*user* => ({

                id: *user*.id,

                username: *user*.username,

                nom: *user*.nom,

                prenom: *user*.prenom,

                cadre: *user*.cadre ? { // CORRECTION: Accédez avec l'alias en minuscule

                    id: *user*.cadre.id,

                    service: *user*.cadre.service,

                    fonction: *user*.cadre.fonction,

                    responsibility\_scope: *user*.cadre.responsibility\_scope,

                    responsible\_escadron\_id: *user*.cadre.responsible\_escadron\_id

                } : null

            }))

        };

        console.log(`[MiseAJourRoutes] GET /daily-updates - Retour de ${response.completed.length} soumissions complétées et ${response.pending.length} en attente.`);

*res*.status(200).json(response);

    } catch (error) {

        console.error('[MiseAJourRoutes] GET /daily-updates - Erreur lors de la récupération des mises à jour quotidiennes :', error);

*res*.status(500).json({ message: 'Erreur serveur lors de la récupération des mises à jour quotidiennes.' });

    }

});

// GET /api/mises-a-jour/daily-stats

router.get('/daily-stats', async (*req*, *res*) => {

    console.log('[MiseAJourRoutes] GET /daily-stats - Récupération des statistiques journalières.');

    try {

        const { date } = *req*.query; // Date au format AAAA-MM-JJ

        if (!date) {

            console.warn('[MiseAJourRoutes] GET /daily-stats - Erreur: Le paramètre "date" est requis.');

            return *res*.status(400).json({ message: 'Le paramètre "date" est requis.' });

        }

        const stats = await HistoriqueStatsJournalieresCadres.findOne({

            where: { date: date }

        });

        console.log(`[MiseAJourRoutes] GET /daily-stats - Statistiques pour la date ${date}: ${stats ? 'Trouvées' : 'Non trouvées'}.`);

        if (!stats) {

            return *res*.status(404).json({ message: 'Statistiques non trouvées pour cette date.' });

        }

*res*.status(200).json(stats);

    } catch (error) {

        console.error('[MiseAJourRoutes] GET /daily-stats - Erreur lors de la récupération des statistiques journalières :', error);

*res*.status(500).json({ message: 'Erreur serveur lors de la récupération des statistiques journalières.' });

    }

});

// POST /api/mises-a-jour/recalculate-stats

router.post('/recalculate-stats', isAdmin, async (*req*, *res*) => {

    console.log('[MiseAJourRoutes] POST /recalculate-stats - Déclenchement du recalcul des statistiques.');

    try {

        const { date } = *req*.body;

        if (!date) {

            console.warn('[MiseAJourRoutes] POST /recalculate-stats - Erreur: Le paramètre "date" est requis.');

            return *res*.status(400).json({ message: 'Le paramètre "date" est requis pour recalculer les statistiques.' });

        }

        console.log(`[MiseAJourRoutes] POST /recalculate-stats - Recalcul des statistiques pour la date: ${date}.`);

        await calculateCurrentAggregateStats(date);

        console.log(`[MiseAJourRoutes] POST /recalculate-stats - Statistiques agrégées recalculées avec succès pour la date ${date}.`);

*res*.status(200).json({ message: `Statistiques agrégées recalculées pour la date ${date}.` });

    } catch (error) {

        console.error('[MiseAJourRoutes] POST /recalculate-stats - Erreur lors du recalcul des statistiques agrégées :', error);

*res*.status(500).json({ message: 'Erreur lors du recalcul des statistiques agrégées.' });

    }

});

// GET /api/mises-a-jour

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

    console.log(`[MiseAJourRoutes] GET / - Récupération de toutes les soumissions pour l'utilisateur ${*req*.user.username} (Rôle: ${*req*.user.role}).`);

    const whereClause = {};

    if (*req*.user.role === 'Standard') {

        try {

            const userWithCadre = await User.findByPk(*req*.user.id, {

                // CORRECTION: Utilisez l'alias 'cadre' (minuscule 'c')

                include: [{ model: Cadre, as: 'cadre', attributes: ['id'] }]

            });

            if (userWithCadre && userWithCadre.cadre) { // CORRECTION: Accédez avec l'alias en minuscule

                whereClause.cadre\_id = userWithCadre.cadre.id; // CORRECTION: Accédez avec l'alias en minuscule

                console.log(`[MiseAJourRoutes] GET / - Utilisateur Standard, filtrage par cadre\_id: ${whereClause.cadre\_id}.`);

            } else {

                console.warn(`[MiseAJourRoutes] GET / - Utilisateur Standard ${*req*.user.username} sans cadre lié, retournant un tableau vide.`);

                return *res*.status(200).json([]);

            }

        } catch (error) {

            console.error('[MiseAJourRoutes] GET / - Erreur lors de la récupération du cadre utilisateur pour le filtrage :', error);

            return *res*.status(500).json({ message: 'Erreur serveur lors de la détermination des permissions de filtrage.' });

        }

    }

    if (*req*.query.status) {

        const allowedStatuses = ['En attente', 'Validée', 'Rejetée'];

        if (allowedStatuses.includes(*req*.query.status)) {

            whereClause.status = *req*.query.status;

            console.log(`[MiseAJourRoutes] GET / - Filtrage par statut: ${*req*.query.status}.`);

        } else {

            console.warn(`[MiseAJourRoutes] GET / - Statut de recherche invalide fourni : ${*req*.query.status}. Ignoré.`);

        }

    }

    if (*req*.query.date) {

        whereClause.update\_date = *req*.query.date;

        console.log(`[MiseAJourRoutes] GET / - Filtrage par date de mise à jour: ${*req*.query.date}.`);

    }

    if (*req*.query.startDate && *req*.query.endDate) {

        whereClause.update\_date = {

            [Op.between]: [*req*.query.startDate, *req*.query.endDate]

        };

        console.log(`[MiseAJourRoutes] GET / - Filtrage par plage de dates: ${*req*.query.startDate} à ${*req*.query.endDate}.`);

    } else if (*req*.query.startDate) {

        whereClause.update\_date = { [Op.gte]: *req*.query.startDate };

        console.log(`[MiseAJourRoutes] GET / - Filtrage par date de début: ${*req*.query.startDate}.`);

    } else if (*req*.query.endDate) {

        whereClause.update\_date = { [Op.lte]: *req*.query.endDate };

        console.log(`[MiseAJourRoutes] GET / - Filtrage par date de fin: ${*req*.query.endDate}.`);

    }

    if (*req*.query.submittedById) {

        whereClause.submitted\_by\_id = *req*.query.submittedById;

        console.log(`[MiseAJourRoutes] GET / - Filtrage par ID du soumetteur: ${*req*.query.submittedById}.`);

    }

    if (*req*.query.cadreId) {

        whereClause.cadre\_id = *req*.query.cadreId;

        console.log(`[MiseAJourRoutes] GET / - Filtrage par ID du cadre: ${*req*.query.cadreId}.`);

    }

    try {

        const soumissions = await MiseAJour.findAll({

            where: whereClause,

            include: [

                { model: User, as: 'SubmittedBy', attributes: ['id', 'username', 'nom', 'prenom'] },

                { model: Cadre, as: 'Cadre', attributes: ['id', 'service', 'fonction'] }, // Maintenez cet alias si c'est celui de MiseAJour

                { model: User, as: 'ValidatedBy', attributes: ['id', 'username', 'nom', 'prenom'], required: false }

            ],

            order: [['update\_date', 'DESC'], ['created\_at', 'DESC']],

        });

        console.log(`[MiseAJourRoutes] GET / - Récupération de ${soumissions.length} soumissions avec succès.`);

        return *res*.status(200).json(soumissions);

    } catch (error) {

        console.error('[MiseAJourRoutes] GET / - Erreur lors de la récupération des soumissions :', error);

        return *res*.status(500).json({ message: 'Erreur serveur lors de la récupération des soumissions.' });

    }

});

// GET /api/mises-a-jour/:id

router.get('/:id', async (*req*, *res*) => {

    const submissionId = *req*.params.id;

    console.log(`[MiseAJourRoutes] GET /:id - Récupération de la soumission ID: ${submissionId}.`);

    try {

        const soumission = await MiseAJour.findByPk(submissionId, {

            include: [

                { model: User, as: 'SubmittedBy', attributes: ['id', 'username', 'nom', 'prenom'] },

                { model: Cadre, as: 'Cadre', attributes: ['id', 'service', 'fonction'] }, // Maintenez cet alias si c'est celui de MiseAJour

                { model: User, as: 'ValidatedBy', attributes: ['id', 'username', 'nom', 'prenom'], required: false }

            ]

        });

        if (!soumission) {

            console.warn(`[MiseAJourRoutes] GET /:id - Soumission ID: ${submissionId} non trouvée.`);

            return *res*.status(404).json({ message: 'Soumission non trouvée.' });

        }

        if (*req*.user.role === 'Standard') {

            console.log(`[MiseAJourRoutes] GET /:id - Utilisateur Standard, vérification des permissions pour soumission ID: ${submissionId}.`);

            try {

                const userWithCadre = await User.findByPk(*req*.user.id, {

                    // CORRECTION: Utilisez l'alias 'cadre' (minuscule 'c')

                    include: [{ model: Cadre, as: 'cadre', attributes: ['id'] }]

                });

                if (!userWithCadre || !userWithCadre.cadre || soumission.cadre\_id !== userWithCadre.cadre.id) { // CORRECTION: Accédez avec l'alias en minuscule

                    console.warn(`[MiseAJourRoutes] GET /:id - Accès non autorisé pour l'utilisateur ${*req*.user.username} (ID: ${*req*.user.id}) à la soumission ID: ${submissionId}.`);

                    return *res*.sendStatus(403);

                }

                console.log(`[MiseAJourRoutes] GET /:id - Accès autorisé pour l'utilisateur ${*req*.user.username} (ID: ${*req*.user.id}) à la soumission ID: ${submissionId}.`);

            } catch (error) {

                console.error('[MiseAJourRoutes] GET /:id - Erreur lors de la vérification du cadre utilisateur pour la permission :', error);

                return *res*.status(500).json({ message: 'Erreur serveur lors de la vérification des permissions.' });

            }

        }

        console.log(`[MiseAJourRoutes] GET /:id - Soumission ID: ${submissionId} récupérée avec succès.`);

        return *res*.status(200).json(soumission);

    } catch (error) {

        console.error('[MiseAJourRoutes] GET /:id - Erreur lors de la récupération de la soumission :', error);

        return *res*.status(500).json({ message: 'Erreur serveur lors de la récupération de la soumission.' });

    }

});

// POST /api/mises-a-jour/:id/validate

router.post('/:id/validate', isAdmin, async (*req*, *res*) => {

    const submissionId = *req*.params.id;

    const { status } = *req*.body;

    console.log(`[MiseAJourRoutes] POST /:id/validate - Tente de valider/rejeter la soumission ID: ${submissionId} avec le statut: ${status}.`);

    const allowedStatuses = ['Validée', 'Rejetée'];

    if (!status || !allowedStatuses.includes(status)) {

        console.warn(`[MiseAJourRoutes] POST /:id/validate - Statut invalide fourni: "${status}".`);

        return *res*.status(400).json({ message: `Le statut ('Validée' ou 'Rejetée') est requis dans le corps de la requête.` });

    }

    try {

        const soumission = await MiseAJour.findByPk(submissionId, {

            include: [

                {

                    model: Cadre,

                    as: 'Cadre', // Maintenez cet alias si c'est celui de MiseAJour

                }

            ]

        });

        if (!soumission) {

            console.warn(`[MiseAJourRoutes] POST /:id/validate - Soumission ID: ${submissionId} non trouvée.`);

            return *res*.status(404).json({ message: 'Soumission non trouvée.' });

        }

        if (soumission.status !== 'En attente') {

            console.warn(`[MiseAJourRoutes] POST /:id/validate - Soumission ID: ${submissionId} a déjà le statut "${soumission.status}".`);

            return *res*.status(400).json({ message: `Cette soumission a déjà le statut "${soumission.status}".` });

        }

        soumission.status = status;

        soumission.validated\_by\_id = *req*.user.id;

        soumission.validation\_date = **new** *Date*();

        await soumission.save();

        console.log(`[MiseAJourRoutes] POST /:id/validate - Soumission ID: ${submissionId} mise à jour au statut "${status}".`);

        if (soumission.status === 'Validée') {

            console.log(`[MiseAJourRoutes] POST /:id/validate - La soumission est "Validée". Déclenchement de la mise à jour des statuts des personnes.`);

            const cadreSoumetteur = soumission.Cadre; // Accédez avec l'alias défini dans MiseAJour (supposé 'Cadre' majuscule ici)

            const dateMiseAJour = soumission.update\_date;

            if (cadreSoumetteur) {

                const scopeType = cadreSoumetteur.responsibility\_scope;

                let cadresToUpdate = [];

                let elevesToUpdate = [];

                console.log(`[MiseAJourRoutes] POST /:id/validate - Scope de responsabilité du cadre: ${scopeType}`);

                if (scopeType === 'Service' && cadreSoumetteur.service) {

                    console.log(`[MiseAJourRoutes] POST /:id/validate - Recherche de cadres pour le service: ${cadreSoumetteur.service}`);

                    cadresToUpdate = await Cadre.findAll({

                        where: { service: cadreSoumetteur.service },

                        attributes: ['id']

                    });

                    console.log(`[MiseAJourRoutes] POST /:id/validate - Trouvé ${cadresToUpdate.length} cadres dans le service.`);

                } else if (scopeType === 'Escadron' && cadreSoumetteur.responsible\_escadron\_id) {

                    console.log(`[MiseAJourRoutes] POST /:id/validate - Recherche de cadres pour l'escadron: ${cadreSoumetteur.responsible\_escadron\_id}`);

                    cadresToUpdate = await Cadre.findAll({

                        where: { responsible\_escadron\_id: cadreSoumetteur.responsible\_escadron\_id },

                        attributes: ['id']

                    });

                    console.log(`[MiseAJourRoutes] POST /:id/validate - Trouvé ${cadresToUpdate.length} cadres dans l'escadron.`);

                    // SI VOUS AVEZ UNE TABLE ELEVE SÉPARÉE, DÉCOMMENTEZ ET AJUSTEZ CELA :

                    // console.log(`[MiseAJourRoutes] POST /:id/validate - Recherche d'élèves pour l'escadron: ${cadreSoumetteur.responsible\_escadron\_id}`);

                    // elevesToUpdate = await Eleve.findAll({

                    //     where: { escadron\_id: cadreSoumetteur.responsible\_escadron\_id },

                    //     attributes: ['id']

                    // });

                    // console.log(`[MiseAJourRoutes] POST /:id/validate - Trouvé ${elevesToUpdate.length} élèves dans l'escadron.`);

                } else {

                    console.warn(`[MiseAJourRoutes] POST /:id/validate - Cadre soumetteur (ID: ${cadreSoumetteur.id}) a un scope de responsabilité "${scopeType}" invalide ou incomplet pour la soumission ${soumission.id}. Skipping status update.`);

                }

                if (cadresToUpdate.length > 0) {

                    const cadreIds = cadresToUpdate.map(*c* => *c*.id);

                    console.log(`[MiseAJourRoutes] POST /:id/validate - Mise à jour des statuts pour les cadres IDs: ${cadreIds.join(', ')}`);

                    await Cadre.update(

                        {

                            statut\_absence: 'Présent',

                            date\_debut\_absence: null,

                            motif\_absence: null,

                            motif\_details: null,

                            timestamp\_derniere\_maj\_statut: **new** *Date*()

                        },

                        {

                            where: { id: { [Op.in]: cadreIds } }

                        }

                    );

                    console.log(`[MiseAJourRoutes] POST /:id/validate - Statuts de ${cadresToUpdate.length} cadres mis à jour à 'Présent' pour l'entité ${scopeType} (${cadreSoumetteur.service || cadreSoumetteur.responsible\_escadron\_id}) à la date ${dateMiseAJour}.`);

                }

                // SI VOUS AVEZ UNE TABLE ELEVE SÉPARÉE, DÉCOMMENTEZ ET AJUSTEZ CELA :

                // if (elevesToUpdate.length > 0) {

                //     const eleveIds = elevesToUpdate.map(e => e.id);

                //     console.log(`[MiseAJourRoutes] POST /:id/validate - Mise à jour des statuts pour les élèves IDs: ${eleveIds.join(', ')}`);

                //     await Eleve.update(

                //         {

                //             statut\_absence: 'Présent',

                //             date\_debut\_absence: null,

                //             motif\_absence: null,

                //             motif\_details: null,

                //             timestamp\_derniere\_maj\_statut: new Date()

                //         },

                //         {

                //             where: { id: { [Op.in]: eleveIds } }

                //         }

                //     );

                //     console.log(`[MiseAJourRoutes] POST /:id/validate - Statuts de ${elevesToUpdate.length} élèves mis à jour à 'Présent' pour l'escadron ${cadreSoumetteur.responsible\_escadron\_id} à la date ${dateMiseAJour}.`);

                // }

            }

        }

        const soumissionValidee = await MiseAJour.findByPk(soumission.id, {

            include: [

                { model: User, as: 'SubmittedBy', attributes: ['id', 'username', 'nom', 'prenom'] },

                { model: Cadre, as: 'Cadre', attributes: ['id', 'service', 'fonction'] }, // Maintenez cet alias si c'est celui de MiseAJour

                { model: User, as: 'ValidatedBy', attributes: ['id', 'username', 'nom', 'prenom'], required: false }

            ]

        });

        console.log(`[MiseAJourRoutes] POST /:id/validate - Réponse finale pour soumission ID: ${submissionId}.`);

        return *res*.status(200).json({ message: `Soumission mise à jour au statut "${soumission.status}"`, soumission: soumissionValidee });

    } catch (error) {

        console.error('[MiseAJourRoutes] POST /:id/validate - Erreur lors de la validation individuelle de la soumission :', error);

        if (error.name === 'SequelizeValidationError') {

            return *res*.status(400).json({ message: 'Erreur de validation : ' + error.errors.map(*e* => *e*.message).join(', '), errors: error.errors });

        }

        return *res*.status(500).json({ message: 'Erreur serveur lors de la validation individuelle de la soumission.' });

    }

});

// POST /api/mises-a-jour/validate-batch

router.post('/validate-batch', isAdmin, async (*req*, *res*) => {

    const { submissionIds, status } = *req*.body;

    console.log(`[MiseAJourRoutes] POST /validate-batch - Tente de valider/rejeter par lot les soumissions IDs: [${submissionIds.join(', ')}] avec le statut: ${status}.`);

    if (!*Array*.isArray(submissionIds) || submissionIds.length === 0) {

        console.warn('[MiseAJourRoutes] POST /validate-batch - Erreur: Liste d\'IDs de soumissions manquante ou vide.');

        return *res*.status(400).json({ message: 'Une liste d\'IDs de soumissions (submissionIds) est requise dans le corps de la requête.' });

    }

    const allowedStatuses = ['Validée', 'Rejetée'];

    if (!status || !allowedStatuses.includes(status)) {

        console.warn(`[MiseAJourRoutes] POST /validate-batch - Statut invalide fourni: "${status}".`);

        return *res*.status(400).json({ message: `Le statut ('Validée' ou 'Rejetée') est requis dans le corps de la requête.` });

    }

    try {

        const soumissions = await MiseAJour.findAll({

            where: {

                id: { [Op.in]: submissionIds },

                status: 'En attente'

            },

            include: [

                {

                    model: User,

                    as: 'SubmittedBy',

                    attributes: ['id', 'username', 'cadre\_id'],

                    include: [

                        {

                            model: Cadre,

                            as: 'cadre', // CORRECTION: Utilisez l'alias 'cadre' (minuscule 'c')

                            // IMPORTANT: Assurez-vous d'inclure 'responsibility\_scope' et 'responsible\_escadron\_id'

                            attributes: ['id', 'service', 'responsibility\_scope', 'responsible\_escadron\_id'],

                        }

                    ]

                }

            ]

        });

        console.log(`[MiseAJourRoutes] POST /validate-batch - Trouvé ${soumissions.length} soumissions en attente parmi celles fournies.`);

        if (soumissions.length === 0) {

            return *res*.status(404).json({ message: 'Aucune soumission en attente trouvée avec les IDs fournis.' });

        }

        const validation\_date = **new** *Date*();

        const validated\_by\_id = *req*.user.id;

        await MiseAJour.update(

            {

                status: status,

                validated\_by\_id: validated\_by\_id,

                validation\_date: validation\_date

            },

            {

                where: {

                    id: { [Op.in]: soumissions.map(*s* => *s*.id) }

                }

            }

        );

        console.log(`[MiseAJourRoutes] POST /validate-batch - ${soumissions.length} soumissions mises à jour au statut "${status}".`);

        if (status === 'Validée') {

            console.log(`[MiseAJourRoutes] POST /validate-batch - Déclenchement de la mise à jour des statuts des personnes pour les soumissions validées.`);

            for (const soumission of soumissions) {

                const soumetteurUser = soumission.SubmittedBy;

                const cadreSoumetteur = soumetteurUser ? soumetteurUser.cadre : null; // CORRECTION: Accédez avec l'alias en minuscule

                const dateMiseAJour = soumission.update\_date;

                if (!cadreSoumetteur) {

                    console.warn(`[MiseAJourRoutes] POST /validate-batch - Soumetteur de la mise à jour (User ID: ${soumetteurUser ? soumetteurUser.id : 'N/A'}) non lié à un Cadre pour la soumission ${soumission.id}. Skipping status update.`);

                    continue;

                }

                const scopeType = cadreSoumetteur.responsibility\_scope;

                let personnesIds = { cadres: [], eleves: [] }; // Initialiser également les élèves

                if (scopeType === 'Service' && cadreSoumetteur.service) {

                    console.log(`[MiseAJourRoutes] POST /validate-batch - Traitement de la soumission pour le Service: ${cadreSoumetteur.service}`);

                    const cadresDuService = await Cadre.findAll({

                        where: { service: cadreSoumetteur.service },

                        attributes: ['id']

                    });

                    personnesIds.cadres = cadresDuService.map(*c* => *c*.id);

                    console.log(`[MiseAJourRoutes] POST /validate-batch - Trouvé ${personnesIds.cadres.length} cadres pour le service.`);

                } else if (scopeType === 'Escadron' && cadreSoumetteur.responsible\_escadron\_id) {

                    // --- NOUVELLE LOGIQUE POUR LES ESCADRONS ---

                    console.log(`[MiseAJourRoutes] POST /validate-batch - Traitement de la soumission pour l'Escadron ID: ${cadreSoumetteur.responsible\_escadron\_id}`);

                    const cadresDeLEscadron = await Cadre.findAll({

                        where: {

                            responsible\_escadron\_id: cadreSoumetteur.responsible\_escadron\_id

                        },

                        attributes: ['id']

                    });

                    personnesIds.cadres = cadresDeLEscadron.map(*c* => *c*.id);

                    console.log(`[MiseAJourRoutes] POST /validate-batch - Trouvé ${personnesIds.cadres.length} cadres pour l'escadron.`);

                } else {

                    console.warn(`[MiseAJourRoutes] POST /validate-batch - Cadre soumetteur (ID: ${cadreSoumetteur.id}) a un scope de responsabilité "${scopeType}" invalide ou incomplet pour la soumission ${soumission.id}. Skipping status update.`);

                    continue;

                }

                // Maintenant, mettez à jour les statuts des cadres concernés

                if (personnesIds.cadres.length > 0) {

                    await Cadre.update(

                        {

                            statut\_absence: 'Présent', // Ou le statut qui convient après validation

                            date\_debut\_absence: null,

                            motif\_absence: null,

                            motif\_details: null,

                            timestamp\_derniere\_maj\_statut: **new** *Date*()

                        },

                        {

                            where: { id: { [Op.in]: personnesIds.cadres } }

                        }

                    );

                    console.log(`[MiseAJourRoutes] POST /validate-batch - Statuts des cadres mis à jour à 'Présent' pour les IDs: ${personnesIds.cadres.join(', ')}.`);

                }

            }

        }

        const soumissionsMisesAJour = await MiseAJour.findAll({

            where: { id: { [Op.in]: soumissions.map(*s* => *s*.id) } },

            include: [

                { model: User, as: 'SubmittedBy', attributes: ['id', 'username'] },

                { model: Cadre, as: 'Cadre', attributes: ['id', 'service', 'fonction', 'responsibility\_scope', 'responsible\_escadron\_id'] }, // Maintenez cet alias

                { model: User, as: 'ValidatedBy', attributes: ['id', 'username'], required: false }

            ]

        });

        console.log(`[MiseAJourRoutes] POST /validate-batch - Opération par lot terminée. Retour de ${soumissionsMisesAJour.length} soumissions mises à jour.`);

        return *res*.status(200).json({

            message: `${soumissionsMisesAJour.length} soumission(s) mise(s) à jour au statut "${status}".`,

            updatedSubmissions: soumissionsMisesAJour

        });

    } catch (error) {

        console.error('[MiseAJourRoutes] POST /validate-batch - Erreur lors de la validation groupée :', error);

        if (error.name === 'SequelizeValidationError') {

            return *res*.status(400).json({ message: 'Erreur de validation : ' + error.errors.map(*e* => *e*.message).join(', '), errors: error.errors });

        }

        return *res*.status(500).json({ message: 'Erreur serveur lors de la validation groupée.' });

    }

});

// GET /api/mises-a-jour/cadres/summary

router.get('/cadres/summary', async (*req*, *res*) => {

    console.log(`[MiseAJourRoutes] GET /cadres/summary - Vérification des permissions pour l'utilisateur ${*req*.user.username} (Rôle: ${*req*.user.role}).`);

    try {

        const date = *req*.query.date; // Date au format 'AAAA-MM-JJ'

        if (!date) {

            console.warn('[MiseAJourRoutes] GET /cadres/summary - Erreur: Le paramètre de date est requis.');

            return *res*.status(400).json({ message: 'Le paramètre de date est requis.' });

        }

        // Déterminer la "journée historique actuelle" en fonction du fuseau horaire de l'application

        const now = **new** *Date*();

        const currentHistoricalDateLabel = getHistoricalDate(now); // Utilisation de votre utilitaire de date

        // --- DEBUT DES LOGS DE DEBUGGING POUR SUMMARY ---

        console.log(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Date demandée par frontend (req.query.date): ${*req*.query.date}`);

        console.log(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Journée historique actuelle (calculée par API via getHistoricalDate): ${currentHistoricalDateLabel}`);

        console.log(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Comparaison: "${req.query.date}" === "${currentHistoricalDateLabel}" ? ${req.query.date === currentHistoricalDateLabel}`);

        // --- FIN DES LOGS DE DEBUGGING POUR SUMMARY ---

        let stats;

        // Si la date demandée est la journée historique actuelle, calculer les stats en temps réel

        if (date === currentHistoricalDateLabel) {

            console.log(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Condition VRAIE: Requête pour la journée historique actuelle (${date}). Calcul des stats en temps réel.`);

            const realTimeStats = await calculateCurrentAggregateStats(); // Appel de la fonction de calcul en temps réel

            stats = {

                total\_cadres: realTimeStats.total,

                absents\_cadres: realTimeStats.absent,

                presents\_cadres: realTimeStats.present,

                indisponibles\_cadres: realTimeStats.indisponible,

                sur\_le\_rang\_cadres: realTimeStats.surLeRang,

                date\_snapshot: date // S'assurer que la date du snapshot est bien la date demandée

            };

            console.log(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Stats en temps réel calculées: Total=${stats.total\_cadres}, Absent=${stats.absents\_cadres}, Indisponible=${stats.indisponibles\_cadres}, Présent=${stats.presents\_cadres}, Sur le rang=${stats.sur\_le\_rang\_cadres}`);

        } else {

            // Sinon, récupérer les stats de la table d'historique (snapshot)

            console.log(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Condition FAUSSE: Requête pour une journée historique passée (${date}). Récupération des stats du snapshot.`);

            stats = await HistoriqueStatsJournalieresCadres.findOne({

                where: { date\_snapshot: date }

            });

            console.log(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Stats Historique trouvées: ${stats ? `Total=${stats.total\_cadres}, Absent=${stats.absents\_cadres}, Indisponible=${stats.indisponibles\_cadres}` : 'Aucune.'}`);

        }

        if (!stats) {

            // Retourner des zéros si aucune statistique n'est trouvée pour la date

            console.warn(`[MiseAJourRoutes] [DEBUG\_SUMMARY] Aucune statistique trouvée pour la date ${date}. Retourne des zéros.`);

            return *res*.status(200).json({

                message: 'Aucune statistique trouvée pour cette date.',

                total\_cadres: 0,

                absents\_cadres: 0,

                presents\_cadres: 0,

                indisponibles\_cadres: 0,

                sur\_le\_rang\_cadres: 0,

                date\_snapshot: date

            });

        }

        console.log(`[MiseAJourRoutes] GET /cadres/summary - Retour des statistiques pour la date ${date}.`);

        return *res*.json(stats);

    } catch (error) {

        console.error('[MiseAJourRoutes] [ERROR\_SUMMARY] Erreur lors de la récupération du résumé des cadres :', error);

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

    }

});

// GET /api/mises-a-jour/users/:userId/submissions

router.get('/users/:userId/submissions', async (*req*, *res*) => {

    const targetUserId = parseInt(*req*.params.userId, 10);

    const dateString = *req*.query.date;

    console.log(`[MiseAJourRoutes] GET /users/:userId/submissions - Récupération des soumissions pour User ID: ${targetUserId} à la date: ${dateString}.`);

    if (isNaN(targetUserId)) {

        console.warn('[MiseAJourRoutes] GET /users/:userId/submissions - Erreur: L\'ID utilisateur dans l\'URL est invalide.');

        return *res*.status(400).json({ message: 'L\'ID utilisateur dans l\'URL est invalide.' });

    }

    if (!dateString) {

        console.warn('[MiseAJourRoutes] GET /users/:userId/submissions - Erreur: Le paramètre "date" est manquant.');

        return *res*.status(400).json({ message: 'Le paramètre "date" est requis dans la chaîne de requête.' });

    }

    if (*req*.user.role !== 'Admin' && *req*.user.id !== targetUserId) {

        console.warn(`[MiseAJourRoutes] GET /users/:userId/submissions - Utilisateur ${*req*.user.username} (ID: ${*req*.user.id}, Rôle: ${*req*.user.role}) a tenté d'accéder aux soumissions de l'utilisateur ${targetUserId} sans permission.`);

        return *res*.status(403).json({ message: 'Vous n\'êtes pas autorisé à voir les soumissions de cet utilisateur.' });

    }

    try {

        const periodStart = getHistoricalDayStartTime(dateString);

        const periodEnd = getHistoricalDayEndTime(dateString);

        console.log(`[MiseAJourRoutes] GET /users/:userId/submissions - Période de recherche (UTC): ${periodStart.toISOString()} à ${periodEnd.toISOString()}`);

        const soumissions = await MiseAJour.findAll({

            where: {

                submitted\_by\_id: targetUserId,

                created\_at: {

                    [Op.between]: [periodStart, periodEnd]

                },

            },

            include: [

                {

                    model: User,

                    as: 'SubmittedBy',

                    attributes: ['id', 'username', 'nom', 'prenom'],

                    include: [

                        {

                            model: Cadre,

                            as: 'cadre', // CORRECTION: Utilisez l'alias 'cadre' (minuscule 'c')

                            attributes: ['id', 'service', 'fonction'],

                            required: false

                        }

                    ]

                },

                {

                    model: User,

                    as: 'ValidatedBy',

                    attributes: ['id', 'username', 'nom', 'prenom'],

                    required: false

                },

            ],

            order: [['created\_at', 'ASC']],

        });

        console.log(`[MiseAJourRoutes] GET /users/:userId/submissions - Récupéré ${soumissions.length} soumissions pour l'utilisateur ${targetUserId}.`);

        return *res*.status(200).json(soumissions);

    } catch (error) {

        console.error(`[MiseAJourRoutes] GET /users/:userId/submissions - Erreur lors de la récupération des soumissions pour l'utilisateur ${targetUserId} à la date ${dateString}:`, error);

        return *res*.status(500).json({ message: 'Erreur serveur lors de la récupération des détails des soumissions.' });

    }

});

*module*.*exports* = router;