const mongoose = require("mongoose");

const Joi = require("joi");

// ✅ Schéma Mongoose

const formationSchema = new mongoose.Schema({

    titre: { type: String, required: true },

    description: { type: String },

    domaine: { type: String, required: true },

    competenceVisee: [{ type: String }],

    dateDebut: { type: Date, required: true },

    dateFin: { type: Date, required: true },

    placeDispo: { type: Number, required: true, min: 0 },

    numeroSalle: { type: String, required: true }

}, { timestamps: true });

const Formation = mongoose.model("Formation", formationSchema);

// ✅ Validation création

function valid\_CreationFormation(obj) {

    const schema = Joi.object({

        titre: Joi.string().min(3).required(),

        description: Joi.string().allow('', null),

        domaine: Joi.string().required(),

        competenceVisee: Joi.array().items(Joi.string()),

        dateDebut: Joi.date().required(),

        dateFin: Joi.date().greater(Joi.ref('dateDebut')).required(),

        placeDispo: Joi.number().min(0).required(),

        numeroSalle: Joi.string().required() // 🔥 Validation ajoutée

    });

    return schema.validate(obj);

}

// ✅ Validation mise à jour (tous les champs optionnels)

function valid\_UpdateFormation(obj) {

    const schema = Joi.object({

        titre: Joi.string().min(3),

        description: Joi.string().allow('', null),

        domaine: Joi.string(),

        competenceVisee: Joi.array().items(Joi.string()),

        dateDebut: Joi.date(),

        dateFin: Joi.date().greater(Joi.ref('dateDebut')),

        placeDispo: Joi.number().min(0),

        numeroSalle: Joi.string() // 🔥 Optionnel pour update

    });

    return schema.validate(obj);

}

// ✅ Validation suppression

function valid\_DeleteFormation(obj) {

    const schema = Joi.object({

        id: Joi.string().required() // ID obligatoire

    });

    return schema.validate(obj);

}

module.exports = {

    Formation,

    valid\_CreationFormation,

    valid\_UpdateFormation,

    valid\_DeleteFormation

};

const {Formation,valid\_CreationFormation,valid\_DeleteFormation,valid\_UpdateFormation} = require("../models/formation");

const asyncHandler = require ("express-async-handler");

const {} = require("../utils/mailer");

require("dotenv").config();

//Get Formation (can be filtred by any filter)

/\*\*

 \* @file formationController.js

 \* @description Controller pour gérer les formations

 \* @access Public

 \* @desc Récupérer toutes les formations avec possibilité de filtres dynamiques

 \* @route GET /api/formations

 \* @returns {Array<Object>}

 \*/

// 👉 GET Formations (avec filtres ou toutes si aucun filtre)

const getFormations = asyncHandler(async (req, res) => {

  const { titre, domaine, dateDebut, dateFin, minPlace, maxPlace } = req.query || {};

  let filtre = {};

  // 🔍 Recherche partielle dans le titre

  if (titre) {

    filtre.titre = { $regex: titre, $options: "i" };

  }

  // 🔍 Recherche partielle dans le domaine

  if (domaine) {

    filtre.domaine = { $regex: domaine, $options: "i" };

  }

  // 📅 Filtrage par intervalle de dates

  if (dateDebut || dateFin) {

    filtre.dateDebut = {};

    if (dateDebut) filtre.dateDebut.$gte = new Date(dateDebut);

    if (dateFin) filtre.dateDebut.$lte = new Date(dateFin);

  }

  // 🎟️ Filtrage par nombre de places disponibles

  if (minPlace || maxPlace) {

    filtre.placeDispo = {};

    if (minPlace) filtre.placeDispo.$gte = parseInt(minPlace);

    if (maxPlace) filtre.placeDispo.$lte = parseInt(maxPlace);

  }

  // ⚡ Si aucun filtre, retourne toutes les formations

  const formations = await Formation.find(filtre);

  res.status(200).json(formations);

});

/\*\*

 \* @desc    Ajouter une nouvelle formation (admin seulement)

 \* @route   POST /api/formations

 \* @access  Admin

 \*/

const addFormation = asyncHandler(async (req, res) => {

  // ✅ Vérification du rôle

  if (req.user.role !== "admin") {

    return res.status(403).json({ message: "Vous n'êtes pas autorisé(e)" });

  }

  // ✅ Validation des données

  const { error } = valid\_CreationFormation(req.body);

  if (error) return res.status(400).json({ message: error.details[0].message });

  // ✅ Création et sauvegarde

  const formation = new Formation({

    titre: req.body.titre,

    description: req.body.description,

    domaine: req.body.domaine,

    competenceVisee: req.body.competenceVisee,

    dateDebut: req.body.dateDebut,

    dateFin: req.body.dateFin,

    placeDispo: req.body.placeDispo,

    numeroSalle: req.body.numeroSalle,

  });

  const createdFormation = await formation.save();

  // ✅ Réponse

  res.status(201).json(createdFormation);

});

/\*\*

 \* @desc    Supprimer une formation par ID (admin seulement)

 \* @route   DELETE /api/formations/:id

 \* @access  Admin

 \*/

const deleteFormation = asyncHandler(async (req, res) => {

  // ✅ Vérification du rôle admin

  if (req.user.role !== "admin") {

    return res.status(403).json({ message: "Vous n'êtes pas autorisé(e)" });

  }

  const formation = await Formation.findById(req.params.id);

  if (!formation) {

    return res.status(404).json({ message: "Formation non trouvée" });

  }

  // ✅ Version compatible Mongoose 6+

  await Formation.deleteOne({ \_id: req.params.id });

  res.status(200).json({ message: "Formation supprimée avec succès" });

});

/\*\*

 \* @desc    Mettre à jour une formation par ID (admin seulement)

 \* @route   PUT /api/formations/:id

 \* @access  Admin

 \*/

const updateFormation = asyncHandler(async (req, res) => {

  // ✅ Vérification du rôle admin

  if (req.user.role !== "admin") {

    return res.status(403).json({ message: "Vous n'êtes pas autorisé(e)" });

  }

  // ✅ Validation des données

  const { error } = valid\_UpdateFormation(req.body);

  if (error) return res.status(400).json({ message: error.details[0].message });

  // ✅ Recherche et mise à jour

  const updatedFormation = await Formation.findByIdAndUpdate(

    req.params.id,

    { $set: req.body },

    { new: true } // renvoie la formation mise à jour

  );

  if (!updatedFormation) {

    return res.status(404).json({ message: "Formation non trouvée" });

  }

  res.status(200).json(updatedFormation);

});

module.exports = {getFormations,addFormation,deleteFormation,updateFormation}

const express = require("express");

const router = express.Router();

const { getFormations , addFormation, deleteFormation, updateFormation } = require("../controllers/FormationController"); // Nom correct

const { verifyToken } = require("../middleware/verifyToken");

// GET Formations avec filtres

router.get("/", getFormations); // ajoute verifyToken si nécessaire

//Post  create neauveau Formation

router.post("/addFormation",verifyToken,addFormation);

//DELETE Formation by id

router.delete("/deleteFormation/:id", verifyToken, deleteFormation);

// PUT update formation par ID

router.put("/updateFormation/:id", verifyToken, updateFormation);

module.exports = router;

const jwt = require("jsonwebtoken");

require("dotenv").config();

function verifyToken(req, res, next) {

    const token = req.query.token || req.headers['x-access-token'];

    if (!token) {

        return res.status(403).json({ message: "Aucun token fourni." });

    }

    try {

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

        req.user = decoded; // Stocke les données décodées dans `req.user`

        next();

    } catch (error) {

        return res.status(401).json({ message: "Token invalide ou expiré." });

    }

}

function generateToken(user) {

    return jwt.sign(

        { message : "login reussie",

          id: user.\_id, email: user.email, role: user.role },

          process.env.JWT\_SECRET\_KEY,

          { expiresIn: "7d" } //expiration dans 7 days

    );

}

module.exports = { verifyToken, generateToken };