User.js :

const mongoose = require("mongoose");

const Joi = require("joi");

// ✅ Schéma Mongoose optimisé

const userSchema = new mongoose.Schema({

    nom: { type: String, required: true, minlength: 5 },

    email: { type: String, required: true, minlength: 5, unique: true, lowercase: true },

    password: { type: String, required: true, minlength: 8 },

    role: { type: String, enum: ['employe', 'rh'], default: 'employe' },

    competence: [String],

    statut: {

        type: String,

        enum: ['en\_attente', 'actif', 'inactif'],

        default: 'en\_attente'

    },

    validationCode: { type: String },

    validationCodeExpires: { type: Date },

    emailVerified: { type: Boolean, default: false }, // Nouveau champ pour vérifier l'email

}, { timestamps: true });

// ✅ Validation pour l'enregistrement

function valid\_Enregistrement(obj) {

    const schema = Joi.object({

        nom: Joi.string().min(5).required(),

        email: Joi.string().email().min(5).required(),

        password: Joi.string().min(8).required(),

        role: Joi.string().valid('employe', 'rh').default('employe')

    });

    return schema.validate(obj);

}

// ✅ Validation pour la connexion

function valid\_connexion(obj) {

    const schema = Joi.object({

        email: Joi.string().email().min(5).required(),

        password: Joi.string().min(8).required()

    });

    return schema.validate(obj);

}

// ✅ Validation pour la mise à jour d'un utilisateur

function valid\_mise\_a\_jour(obj) {

    const schema = Joi.object({

        nom: Joi.string().min(5),

        email: Joi.string().email().min(5),

        password: Joi.string().min(8),

        role: Joi.string().valid('employe', 'rh'),

        statut: Joi.string().valid('en\_attente', 'actif', 'inactif')

    });

    return schema.validate(obj);

}

// ✅ Fonction simplifiée pour valider un compte utilisateur

function valid\_activation\_compte(obj) {

    const schema = Joi.object({

        statut: Joi.string().valid('actif', 'inactif').required()

    });

    return schema.validate(obj);

}

//✅ Fonction simplifiée  pour valider le code de comfirmation

function valid\_code\_validation(obj){

    const schema = Joi.object({

        email: Joi.string().email().min(5).required(),

        code: Joi.string().length(6).required(),

    });

    return  schema.validate(obj);

}

const User = mongoose.model("User", userSchema);

module.exports = {

    User,

    valid\_Enregistrement,

    valid\_connexion,

    valid\_mise\_a\_jour,

    valid\_activation\_compte,valid\_code\_validation

};

updateController.js

const { User, valid\_mise\_a\_jour, valid\_code\_validation,valid\_activation\_compte } = require("../models/user");

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

const bcrypt = require("bcrypt");

require("dotenv").config();

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

const { sendValidationUpdate, sendValidationConfirmation ,sendStatusActif} = require("../utils/mailer");

// PUT Update user

/\*\*

 \* @file updateController.js

 \* @description Met à jour les informations d'un utilisateur après vérification du token

 \* @route PUT /api/userUpdate/Update/:id

 \* @access Privé - Seul l'utilisateur propriétaire du compte

 \*/

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

    console.log("Decoded user :", req.user);

    //🔐 Vérification le role

    if (req.body.role && req.body.role !== user.role) {

    return res.status(403).json({

        message: "La modification du rôle n'est pas autorisée"

    });

}

    // 🔐 Vérification que l'utilisateur modifie bien son propre compte

    if (req.user.id !== req.params.id) {

        return res.status(403).json({ message: "Modification interdite" });

    }

    // ✅ Validation des données avec Joi

    const { error } = valid\_mise\_a\_jour(req.body);

    if (error) {

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

    }

    // 🔍 Recherche de l'utilisateur dans la base de données

    const user = await User.findById(req.params.id);

    if (!user) {

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

    }

    // 📧 Gestion spéciale pour le changement d'email

    if (req.body.email && req.body.email !== user.email) {

        // Génération d'un code de validation à 6 chiffres

        const validationCode = Math.floor(100000 + Math.random() \* 900000).toString();

        // Expiration du code dans 24 heures

        const validationCodeExpires = new Date(Date.now() + 24 \* 60 \* 60 \* 1000);

        // Stockage temporaire du nouvel email et du code

        user.newEmail = req.body.email; // Champ temporaire pour le nouvel email

        user.validationCode = validationCode;

        user.validationCodeExpires = validationCodeExpires;

        user.emailVerified = false; // Email non validé tant que le code n'est pas confirmé

        await user.save();

        // 📨 Envoi du code de validation par email

        await sendValidationUpdate(req.body.email, validationCode);

        return res.status(200).json({

            message: "Un code de validation a été envoyé à votre nouvelle adresse email.",

            requiresValidation: true

        });

    }

    // 🔐 Hashage du mot de passe s'il est modifié

    if (req.body.password) {

        const salt = await bcrypt.genSalt(10);

        req.body.password = await bcrypt.hash(req.body.password, salt);

    }

    // 📋 Préparation des données de mise à jour (on exclut l'email qui est géré séparément)

    const updateData = { ...req.body };

    delete updateData.email; // L'email est géré via le processus de validation

    // 💾 Mise à jour de l'utilisateur dans la base de données

    const updatedUser = await User.findByIdAndUpdate(

        req.params.id,

        updateData,

        { new: true } // Retourne le document mis à jour

    ).select("-password"); // Exclut le mot de passe de la réponse

    // ✅ Réponse de succès

    res.status(200).json({

        message: "Utilisateur mis à jour avec succès",

        user: updatedUser

    });

});

// POST Validate email update

/\*\*

 \* @file updateController.js

 \* @description Valide la mise à jour de l'adresse email avec le code reçu

 \* @route POST /api/userUpdate/validateEmail

 \* @access Public - Nécessite le code de validation

 \*/

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

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

    if (error) {

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

    }

    // 🔍 Chercher l'utilisateur par le code de validation uniquement

    const user = await User.findOne({ validationCode: req.body.code });

    if (!user) {

        return res.status(404).json({ message: "Code de validation invalide ou expiré" });

    }

    // ⏰ Vérifier si le code n'a pas expiré

    if (user.validationCodeExpires < new Date()) {

        return res.status(400).json({ message: "Le code de validation a expiré" });

    }

    // ✅ Mettre à jour l'email avec celui fourni dans la requête

    user.email = req.body.email;

    user.validationCode = undefined;

    user.validationCodeExpires = undefined;

    user.emailVerified = true;

    await user.save();

    await sendValidationConfirmation(user.email);

    res.status(200).json({

        message: "Email validé et mis à jour avec succès"

    });

});

// PUT Changer le status

/\*\*

 \* @file updateController.js

 \* @description Valide la mise à jour de l'adresse email avec le code reçu

 \* @route  /api/userUpdate/Change\_status

 \* @access only RH (RH)

 \*/

const Change\_status = asyncHandler(async(req, res) => {

    console.log("Decoded user :", req.user);

    // 1. Vérifier que seul un RH peut faire cette action

    if (req.user.role !== "rh") {

        return res.status(403).json({ message: "Non autorisé - Réservé aux RH" });

    }

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

    if (error) {

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

    }

    // 2. Trouver l'utilisateur à modifier

    const user = await User.findById(req.params.id);

    if (!user) {

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

    }

    // 3. Vérifier qu'on ne modifie pas son propre statut

    if (req.user.id === req.params.id) {

        return res.status(403).json({ message: "Vous ne pouvez pas modifier votre propre statut" });

    }

    // 4. Vérifier si le statut est déjà celui demandé (évite opération inutile)

    if (user.statut === req.body.statut) {

        return res.status(200).json({

            message: `Le statut est déjà "${user.statut}" - Aucune modification nécessaire`

        });

    }

    // 5. Mettre à jour le statut

    user.statut = req.body.statut; // ⚠️ Correction: 'statut' pas 'role'

    await user.save();

    // 6. Envoyer l'email de notification

    await sendStatusActif(user.email, user.statut);

    return res.status(200).json({

        message: `Statut modifié avec succès: ${user.statut}`

    });

});

module.exports = { UpdateUser, ValidateEmailUpdate , Change\_status };

updateRoute.js ù

const express = require(

"express");

const router = express.Router();

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

const { UpdateUser, ValidateEmailUpdate ,Change\_status} = require("../controllers/updateController");

// Mise à jour du profil utilisateur (protégée par token)

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

// Validation du code pour la modification d'email

router.post("/validate-email", ValidateEmailUpdate);

// Route pour changer le statut (RH seulement)

router.put("/change-status/:id", verifyToken, Change\_status);

module.exports = router;

mailer.js

const nodemailer = require("nodemailer");

require("dotenv").config();

const transporter = nodemailer.createTransport({

    service: "gmail",

    auth: {

        user: process.env.EMAIL\_USER,

        pass: process.env.EMAIL\_PASS,

    },

});

    //Email de Confirmation de validation du update

    async function sendValidationConfirmation(email){

        const mailOptions = {

            from: process.env.EMAIL\_USER,

            to: email,

            subject: "Votre email a été validé avec succès",

            text: `

                Validation d'email réussie !

                Bonjour,

                Votre adresse email a été validée avec succès et est maintenant activée sur votre compte.

                Vous pouvez désormais utiliser cette adresse email pour vous connecter à votre compte et recevoir toutes nos communications.

                ✅ Email validé avec succès

                Date de validation: ${new Date().toLocaleString('fr-FR')}

                Si vous n'êtes pas à l'origine de cette action, veuillez contacter immédiatement le support.

                Cordialement,

                L'équipe de support`

            };

            return transporter.sendMail(mailOptions);

    }

   //Email de Confirmation de

async function sendStatusActif(email, status) {

    const subject = status === 'actif'

        ? "Votre compte a été activé"

        : "Votre compte a été désactivé";

    const text = status === 'actif'

        ? `Bonjour, votre compte a été activé avec succès. Vous pouvez maintenant vous connecter.`

        : `Bonjour, votre compte a été désactivé. Contactez l'administration pour plus d'informations.`;

    const mailOptions = {

        from: process.env.EMAIL\_USER,

        to: email,

        subject: subject,

        text: text

    };

    return transporter.sendMail(mailOptions);

verify token. Js

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 };