// routes/auth.js

const express = require("express");

const User = require("../models/signup");

const bcrypt = require("bcryptjs");

const jwt = require("jsonwebtoken");

const cookieParser = require("cookie-parser");

require("dotenv").config();

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

const router = express.Router();

router.use(cookieParser());

console.log("🚀 Auth routes loaded");

// Login Route

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

    console.log("🔗 POST /login - Handling login request...");

    try {

        const { email, password } = req.body;

        console.log("📧 Email:", email);

        console.log("🔑 Password:", password);

        if (!email || !password) {

            console.log("❌ All fields are required.");

            return res.status(400).json({ message: "All fields are required" });

        }

        console.log("🔍 Searching for user in database...");

        const user = await User.findOne({ email });

        if (!user) {

            console.log("❌ User not found.");

            return res.status(401).json({ message: "Invalid email or password" });

        }

        console.log("🔍 Comparing passwords...");

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

        if (!isMatch) {

            console.log("❌ Passwords do not match.");

            return res.status(401).json({ message: "Invalid email or password" });

        }

        console.log("🔑 Generating JWT token...");

        const token = jwt.sign({ userId: user.\_id, email: user.email }, process.env.JWT\_SECRET, { expiresIn: "1h" });

        // Set the token in a cookie

        res.cookie('token', token, { httpOnly: true, secure: process.env.NODE\_ENV === 'production' });

        console.log("✅ Login successful.");

        console.log("🔑 Token:", token);

        console.log("👤 User data:", { userId: user.\_id, email: user.email, role: user.role });

        // Return token and user data

        res.json({

            message: "Login successful",

            token,

            user: { userId: user.\_id, email: user.email, role: user.role },

            redirect: "/dashboard"

        });

    } catch (error) {

        console.error("❌ Login Error:", error);

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

    }

});

// Check Authentication Status

router.get("/status", async (req, res) => {

    console.log("🔗 GET /status - Checking authentication status...");

    const authHeader = req.headers.authorization;

    if (!authHeader) {

        console.log("❌ No authorization header found.");

        return res.json({ isAuthenticated: false });

    }

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

    console.log("🔑 Token from header:", token);

    try {

        console.log("🔍 Verifying token...");

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

        console.log("✅ Token verified successfully.");

        console.log("👤 Decoded user data:", decoded);

        // Fetch user data from the database

        const user = await User.findById(decoded.userId);

        if (!user) {

            console.log("❌ User not found in the database.");

            return res.json({ isAuthenticated: false });

        }

        console.log("👤 User data from database:", user);

        // Return user data including the name

        res.json({

            isAuthenticated: true,

            user: {

                userId: user.\_id,

                email: user.email,

                yourname: user.yourname, // Include the user's name

                role: user.role

            }

        });

    } catch (error) {

        console.error("❌ Token verification failed:", error.message);

        res.json({ isAuthenticated: false });

    }

});

// Logout Route

router.post('/logout', (req, res) => {

    console.log("🔗 POST /logout - Handling logout request...");

    res.clearCookie('token');

    console.log("✅ Logout successful. Token cleared.");

    res.json({ message: "Logout successful" });

});

module.exports = router;