const bcrypt = require("bcryptjs");

const jwt = require("jsonwebtoken");

const userSchema = require("../schemas/userModel");

const propertySchema = require("../schemas/propertyModel");

const bookingSchema = require("../schemas/bookingModel");

//////////for registering/////////////////////////////

const registerController = async (req, res) => {

try {

let granted = "";

const existsUser = await userSchema.findOne({ email: req.body.email });

if (existsUser) {

return res

.status(200)

.send({ message: "User already exists", success: false });

}

const password = req.body.password;

const salt = await bcrypt.genSalt(10);

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

req.body.password = hashedPassword;

if (req.body.type === "Owner") {

granted = "ungranted";

const newUser = new userSchema({ ...req.body, granted });

await newUser.save();

} else {

const newUser = new userSchema(req.body);

await newUser.save();

}

///////////aur you can do this////////

// if (req.body.type === "Owner") {

// newUser.set("granted", "pending", { strict: false });

// }

//////////////////// for this, then you need to remove strict keyword from schema//////////////////////

return res.status(201).send({ message: "Register Success", success: true });

} catch (error) {

console.log(error);

return res

.status(500)

.send({ success: false, message: `${error.message}` });

}

};

////for the login

const loginController = async (req, res) => {

try {

const user = await userSchema.findOne({ email: req.body.email });

if (!user) {

return res

.status(200)

.send({ message: "User not found", success: false });

}

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

if (!isMatch) {

return res

.status(200)

.send({ message: "Invalid email or password", success: false });

}

const token = jwt.sign({ id: user.\_id }, process.env.JWT\_KEY, {

expiresIn: "1d",

});

user.password = undefined;

return res.status(200).send({

message: "Login success successfully",

success: true,

token,

user: user,

});

} catch (error) {

console.log(error);

return res

.status(500)

.send({ success: false, message: `${error.message}` });

}

};

/////forgotting password

const forgotPasswordController = async (req, res) => {

try {

const { email, password } = req.body;

// Hash the new password

const salt = await bcrypt.genSalt(10);

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

const updatedUser = await userSchema.findOneAndUpdate(

{ email },

{ password: hashedPassword },

{ new: true }

);

if (!updatedUser) {

return res

.status(200)

.send({ message: "User not found", success: false });

}

await updatedUser.save();

return res.status(200).send({

message: "Password changed successfully",

success: true,

});

} catch (error) {

console.log(error);

return res

.status(500)

.send({ success: false, message: `${error.message}` });

}

};

////auth controller

const authController = async (req, res) => {

console.log(req.body);

try {

const user = await userSchema.findOne({ \_id: req.body.userId });

console.log(user);

if (!user) {

return res

.status(200)

.send({ message: "user not found", success: false });

} else {

return res.status(200).send({

success: true,

data: user,

});

}

} catch (error) {

console.log(error);

return res

.status(500)

.send({ message: "auth error", success: false, error });

}

};

/////////get all properties in home

const getAllPropertiesController = async (req, res) => {

try {

const allProperties = await propertySchema.find({});

if (!allProperties) {

throw new Error("No properties available");

} else {

res.status(200).send({ success: true, data: allProperties });

}

} catch (error) {

console.log(error);

return res

.status(500)

.send({ message: "auth error", success: false, error });

}

};

///////////booking handle///////////////

const bookingHandleController = async (req, res) => {

const { propertyid } = req.params;

const { userDetails, status, userId, ownerId } = req.body;

try {

const booking = new bookingSchema({

propertyId: propertyid,

userID: userId,

ownerID: ownerId,

userName: userDetails.fullName,

phone: userDetails.phone,

bookingStatus: status,

});

await booking.save();

return res

.status(200)

.send({ success: true, message: "Booking status updated" });

} catch (error) {

console.error("Error handling booking:", error);

return res

.status(500)

.send({ success: false, message: "Error handling booking" });

}

};

/////get all bookings for sing tenents//////

const getAllBookingsController = async (req, res) => {

const { userId } = req.body;

try {

const getAllBookings = await bookingSchema.find();

const updatedBookings = getAllBookings.filter(

(booking) => booking.userID.toString() === userId

);

return res.status(200).send({

success: true,

data: updatedBookings,

});

} catch (error) {

console.error(error);

return res

.status(500)

.send({ message: "Internal server error", success: false });

}

};

module.exports = {

registerController,

loginController,

forgotPasswordController,

authController,

getAllPropertiesController,

bookingHandleController,

getAllBookingsController,

};