server.js

const express = require('express');

const mongoose = require('mongoose');

const dotenv = require('dotenv');

const authRoutes = require('./routes/route');

dotenv.config();

const app = express();

app.use(express.json());

// Database connection

mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log('MongoDB connected successfully'))

.catch((error) => console.error('MongoDB connection error:', error));

// Use routes

app.use('/main', authRoutes);

// Start server

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

.env

PORT=4000

EMAIL\_USER=email@gmail.com

EMAIL\_PASS=password

MONGODB\_URI=mongodb://localhost:27017/mydatabase2

model.js

const { default: mongoose } = require("mongoose");

const userschema = new mongoose.Schema({

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

email: { type: String, required: true, unique: true },

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

isVerified: { type: Boolean, default: false }

})

const User = mongoose.model('UserDetails', userschema);

module.exports = User;

token.js

const mongoose = require("mongoose");

// Token schema for storing user verification tokens

const tokenschema = new mongoose.Schema({

userId: {

type: mongoose.Schema.Types.ObjectId,

ref: 'UserDetails', // Reference to the 'UserDetails' model

required: true

},

token: {

type: String,

required: true

},

createdAt: {

type: Date,

default: Date.now,

expires: 3600

} // Token will expire in 1 hour (3600 seconds)

});

const Token = mongoose.model("Token", tokenschema);

module.exports = Token;

route.js

const express = require('express');

const bcrypt = require('bcrypt');

const crypto = require('crypto');

const nodemailer = require('nodemailer');

const User = require('../model/model'); // Ensure path is correct

const Token = require('../model/toke'); // Ensure path is correct

const route = express.Router();

route.post('/register', async (req, res) => {

try {

const { username, email, password } = req.body;

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

if (existingUser) {

return res.status(400).json({ error: existingUser.isVerified ? 'Email already exists' : 'Email already exists. Verification pending.' });

}

// Hash password

const hashedPassword = await bcrypt.hash(password, 10);

// Create new user

const newUser = new User({

username,

email,

password: hashedPassword,

isVerified: false

});

await newUser.save();

// Generate verification token

const verificationToken = crypto.randomBytes(32).toString('hex');

const newToken = new Token({

userId: newUser.\_id,

token: verificationToken,

createdAt: Date.now()

});

await newToken.save();

// Verification URL

const verificationUrl = `${req.protocol}://${req.get('host')}/main/verify/${newUser.\_id}/${verificationToken}`;

// Mail transporter setup

const transporter = nodemailer.createTransport({

service: "gmail",

auth: {

user: process.env.EMAIL\_USER,

pass: process.env.EMAIL\_PASS,

}

});

// Email content

const mailOptions = {

from: process.env.EMAIL\_USER,

to: newUser.email,

subject: 'Email Verification',

text: `Please verify your email by clicking on the following link: ${verificationUrl}`,

html: `<p>Please verify your email by clicking on the following link:</p><a href="${verificationUrl}">${verificationUrl}</a>`

};

try {

await transporter.sendMail(mailOptions);

res.status(201).json({ message: 'User registration successful. Verification email sent.', newUser });

} catch (error) {

console.error('Error sending email:', error.message);

res.status(500).json({ error: `Error sending verification email: ${error.message}` });

}

} catch (error) {

console.error(error);

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

}

});

route.get('/verify/:userId/:token', async (req, res) => {

try {

const { userId, token } = req.params;

console.log(`Received verification request for userId: ${userId} with token: ${token}`);

// Check if token exists

const tokenRecord = await Token.findOne({ userId, token });

if (!tokenRecord) {

console.error('Token not found or expired.');

return res.status(400).json({ error: 'Invalid or expired token' });

}

console.log('Token found:', tokenRecord);

// Find user by userId

const user = await User.findById(userId);

if (!user) {

console.error('User not found.');

return res.status(404).json({ error: 'User not found' });

}

// Set user as verified

user.isVerified = true;

await user.save();

console.log(`User ${user.username} is now verified:`, user.isVerified);

// Delete the token after successful verification

await Token.deleteOne({ \_id: tokenRecord.\_id });

console.log('Token deleted after successful verification.');

res.status(200).json({ message: 'Email verified successfully' });

} catch (error) {

console.error(`Error during email verification: ${error.message}`);

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

}

});

module.exports = route;