const express = require('express');

const mongoose = require('mongoose');

const Pusher = require('pusher');

const cors = require('cors');

require('dotenv').config();

const path = require('path');

const app = express();

app.use(express.json());

app.use(cors());

// Serve static files from the root directory

app.use(express.static(path.join(\_\_dirname)));

// MongoDB connection

const uri = process.env.MONGODB\_URI || "mongodb+srv://chatuser:securepassword123@chatroulette-lite-clust.amhu0.mongodb.net/?retryWrites=true&w=majority&appName=chatroulette-lite-cluster";

mongoose.connect(uri, {

serverSelectionTimeoutMS: 30000, // Increase timeout to 30 seconds

heartbeatFrequencyMS: 10000 // Check connection every 10 seconds

})

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

.catch(err => {

console.error('MongoDB connection error:', err);

process.exit(1);

});

// Pusher setup

const pusher = new Pusher({

appId: process.env.PUSHER\_APP\_ID,

key: process.env.PUSHER\_KEY,

secret: process.env.PUSHER\_SECRET,

cluster: process.env.PUSHER\_CLUSTER,

useTLS: true

});

const User = require('./models/User');

// In-memory store for recent messages (simple deduplication, could be replaced with MongoDB)

const recentMessages = new Set();

const Message = mongoose.model('Message', new mongoose.Schema({

text: String,

from: String,

to: String,

messageId: String,

timestamp: { type: Date, default: Date.now }

}));

app.get('/api/users/nearby', async (req, res) => {

try {

const { latitude, longitude, radius } = req.query;

const lat = parseFloat(latitude);

const lon = parseFloat(longitude);

const maxDistance = parseFloat(radius);

console.log('Query parameters:', { latitude: lat, longitude: lon, radius: maxDistance });

if (isNaN(lat) || isNaN(lon) || isNaN(maxDistance)) {

throw new Error('Invalid latitude, longitude, or radius');

}

const users = await User.aggregate([

{

$geoNear: {

near: { type: 'Point', coordinates: [lon, lat] },

distanceField: 'dist.calculated',

maxDistance: maxDistance,

spherical: true

}

},

{

$project: {

\_id: 1,

preferredName: 1,

preferences: 1,

location: 1,

dist: 1,

online: 1

}

}

]);

console.log('Aggregation result:', users);

const formattedUsers = users.map(user => ({

\_id: user.\_id,

preferredName: user.preferredName || 'Anonymous',

preferences: user.preferences || { topics: ['chat'], ageRange: { min: 18, max: 99 } },

location: user.location || { type: 'Point', coordinates: [0, 0] },

dist: user.dist || { calculated: 0 },

online: user.online || false

}));

console.log('Formatted nearby users:', formattedUsers);

res.json(formattedUsers);

} catch (error) {

console.error('Error fetching nearby users:', error);

res.status(500).json({ error: 'Error fetching nearby users', details: error.message });

}

});

app.delete('/api/user/location', async (req, res) => {

await User.deleteMany({});

pusher.trigger('presence-channel', 'user-location-revoked', {});

res.json({ success: true });

});

app.get('/api/user/dashboard', async (req, res) => {

const userId = req.query.userId;

try {

const user = await User.findById(userId);

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

res.json({

locationHistory: user.location,

onlineStatus: user.online,

preferences: user.preferences,

sharedAt: user.createdAt

});

} catch (error) {

res.status(500).json({ error: 'Error fetching dashboard data' });

}

});

app.post('/api/user/update-preferences', async (req, res) => {

const { userId, preferences, preferredName } = req.body;

// Update user with preferredName and preferences

// Example: await User.findByIdAndUpdate(userId, { preferredName, preferences });

res.json({ success: true, message: 'Preferences updated' });

});

app.post('/message', async (req, res) => {

const { text, from, to, messageId } = req.body;

console.log('Received message:', { text, from, to, messageId });

// Deduplication check

if (messageId && recentMessages.has(messageId)) {

console.log('Duplicate message detected, skipping:', messageId);

return res.sendStatus(200);

}

try {

// Store message in MongoDB

await Message.create({ text, from, to, messageId });

// Trigger Pusher event

pusher.trigger('chat-channel', 'message', { text, from, to, messageId }, (error) => {

if (error) {

console.error('Pusher trigger error:', error);

return res.status(500).json({ error: 'Failed to trigger Pusher event' });

}

});

// Add to recent messages set for deduplication

if (messageId) recentMessages.add(messageId);

// Clean up old messages (keep last 100 unique messages)

if (recentMessages.size > 100) {

const iterator = recentMessages.values();

for (let i = 0; i < recentMessages.size - 100; i++) {

recentMessages.delete(iterator.next().value);

}

}

res.sendStatus(200);

} catch (error) {

console.error('Error processing message:', error.stack);

res.status(500).json({ error: 'Error processing message', details: error.message });

}

});

app.post('/api/user/toggle-location', async (req, res) => {

const { userId, shareLocation } = req.body;

try {

const updatedUser = await User.findByIdAndUpdate(userId, { location: shareLocation ? user.location : null }, { new: true });

res.json(updatedUser);

} catch (error) {

res.status(500).json({ error: 'Error toggling location sharing' });

}

});

app.post('/api/connect/:userId', async (req, res) => {

try {

const { userId } = req.params;

const { preferredName } = req.body;

if (!preferredName) {

return res.status(400).json({ error: 'Preferred name is required' });

}

// Update or create user with preferredName and default location

const user = await User.findByIdAndUpdate(

userId,

{

preferredName,

location: { type: 'Point', coordinates: [0, 0] }, // Default location

preferences: { topics: ['chat'], ageRange: { min: 18, max: 99 } }, // Default preferences

online: true

},

{ new: true, upsert: true }

);

// Simulate finding another user to connect with

const otherUser = await User.findOne({ \_id: { $ne: userId } });

res.json({

success: true,

otherUserName: otherUser ? otherUser.preferredName || 'Anonymous' : 'Anonymous',

preferredName: user.preferredName

});

} catch (error) {

console.error('Error connecting:', error);

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

}

});

app.post('/pusher/auth', (req, res) => {

const socketId = req.body.socket\_id;

const channelName = req.body.channel\_name;

const userId = req.body.user\_id || 'anonymous'; // You can pass userId from the client if needed

const presenceData = {

user\_id: userId,

user\_info: { name: req.body.preferredName || 'Anonymous' }

};

try {

const authResponse = pusher.authenticate(socketId, channelName, presenceData);

res.json(authResponse);

} catch (error) {

console.error('Pusher auth error:', error);

res.status(500).json({ error: 'Pusher authentication failed' });

}

});

// Handle root route to serve index.html

app.get('/', (req, res) => {

res.sendFile(path.join(\_\_dirname, 'index.html'));

});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));