**Project Advanced Software 2025**

* **Apis**
* exportSponsorsToSheet.js
* const { google } = require('googleapis');
* const path = require('path');
* const Sponsor = require('../models/sponsorModel');
* const SCOPES = ['https://www.googleapis.com/auth/spreadsheets'];
* const SERVICE\_ACCOUNT\_FILE = path.join(\_\_dirname, '../config/service-account.json');
* const spreadsheetId = '1LyeTOcVIj1\_58-1lWjqw8kZd1W7PN97Vq2FjTqYr8NY';
* async function exportSponsors() {
* console.log('exportSponsors started');
* try {
* const auth = new google.auth.GoogleAuth({
* keyFile: SERVICE\_ACCOUNT\_FILE,
* scopes: SCOPES,
* });
* const sheets = google.sheets({ version: 'v4', auth });
* const sponsors = await Sponsor.getAll();
* const values = [
* ['ID', 'Name', 'Email', 'Phone', 'Location', 'Status'],
* ...sponsors.map(s => [
* s.id,
* s.name,
* s.email,
* s.phone,
* s.location,
* s.status,
* ]),
* ];
* const resource = { values };
* console.log(values);
* await sheets.spreadsheets.values.update({
* spreadsheetId,
* range: 'Sheet1!A1',
* valueInputOption: 'RAW',
* resource,
* });
* return { success: true, count: sponsors.length };
* } catch (error) {
* return { success: false, error: error.message };
* }
* }
* module.exports = { exportSponsors };
* **config**
* db.js
* const mysql = require('mysql');
* const db = mysql.createConnection({
* host: 'localhost',
* user: 'root',
* password: '',
* database: 'hopeconnect\_db'
* });
* db.connect((err) => {
* if (err) {
* console.error('Failed to connect to the database:', err.message);
* return;
* }
* console.log('Successfully connected to the database');
* });
* module.exports = db;
* **service-account.json**
* {
* "type": "service\_account",
* "project\_id": "my-sheets-api-project-460216",
* "private\_key\_id": "a78d80dacf80f243c531f5936ac668d5cc2f3f4d",
* "private\_key": "-----BEGIN PRIVATE KEY-----\nMIIEuwIBADANBgkqhkiG9w0BAQEFAASCBKUwggShAgEAAoIBAQDCYsZuFknVz49b\nRrh4Dcmqh0NoZ8VIycHxD4SFOXB0TZm3ac5zYUH+S8K1sJPUku10RpF7x88TL9Xn\nqh2/jCykW0uT7m3q1DakRtZd8Tg/Dzm6NkqeUJzpNhzmi6wtKAF0HgrnspEtGnzn\nkxcvzLrSU2o7w5RpgZop/kzWZTYN9AAXmmPq1x8ZAckqF1fRr+0VtglN5MnmT7iM\nQdRf6UBx1rEf6oN9hZcxHUJAWsTODiaR8L/BOWrefIlQwqeoEqkESRvHzGdLI3fw\nMcUebozwyaZdTu2oa4XhP599QuAJMZFu/SJb9Ff/LqGqPwFHM6cLAwAh//biwNCm\no6OYfn2VAgMBAAECgf8neiRwq3M3YbpSyDrq/OaBNsQMcmhttgA2grYVsM7Pr0/s\nnnIZljTlyvFuvO2vToSubrRw5bX3bml5ZCNlVfQ0Cxn4JqThlWYhukEr+DVV1bF2\nOXkqg+DXvTFGnFhBqZF4HS124LNjSV1DFZCm6lE+j62axLTfYA6wPBNQz7rrxmAq\n1bmvQr8tdeQ8hUBYVBJrs1VXQ+mjAeVqtB8xGw9vUBqEwk63Li3B3AZZB6uR4x01\ndu6M2yQub/nzMGaBN3XGuSomtjr36t6tD5BXVwP4M+odZ58AVpalRR3gTV4sgBOi\nY0kys4kes7Erc/fFma4F0OcuU3ETIv9h3Tq29AcCgYEA/8nU/DmL6tExQIad3lCf\n19nUSmTmHTisNRrSLM1NSQ+5RtSOtBVG4UjTBttgXd4z8SzMZNu8O32UKJ78MjaQ\n4y2kNdoADkwVdRuB9wu96ofRTsJlqC71RnopNoF7BQQDL4L7er5TMKm3EISyPT3u\ndBtznRBUUNvKgMZY13jwDacCgYEAwovwoziXdc3YElhNfPEGgnATqK7POw9B/eKZ\nLvVx2zTyV9TJTnsEAnImsF6TcrXezv8v7WwyNCm64fFf7VluEj07xaVaYdKA/4F0\np+FjLw3Jgk/LGmHkmBz8RmxEQlEqpGdTBhct5uec1x3aPpv7OEiR2zpaBf7r7cvY\nPHLP2mMCgYEAwo0KIVnH0S+RrYl2U9yw5yV4caky2XD8klu3FnqE2RhLjNMpQ1GX\nmsygLWv4bQ0QJtUzFVeIYRtP8GGUYsDcDN7vMkKAnl9CRtNIU9YrelIZYjU7yuME\njblm1V1Aa300yxNuIVrXvYCoi7jSNQCA+PDUS9daujGiYbX7ZtBGjasCgYAWqJfZ\nWtEBLn1Y/aHgyG26qXUQs3N9bHSCLm8q8flVz5/eLPFZS6iW5FavOgSwWpBdyYj6\nfRyHyMh0YlmK2lygkTMx/OU++XZ+y8QbfXvA0jlDDSoZo4ctVErppNh6ZNuzlDku\nP2rgNt+kilyN6DrlPkaGoxAzHJK7+n7v42tkAwKBgF5s+jxm1t83Jx7sA+SSFjNO\nDm22JLGz7bzi0bF2MMJqm/uF0erSxUQPPF0ylNwsSllp/DntlNl8OHvdp+IB18DE\nNb8wfC8ksSh2g/6K2MHkZnaZgx1aAwuSEMrMiOoelJiZd+eH+m47dSErQ1Rav8Uf\nFJgY5fPWIiqP6QliwqMv\n-----END PRIVATE KEY-----\n",
* "client\_email": "bayan-futyan@my-sheets-api-project-460216.iam.gserviceaccount.com",
* "client\_id": "105837981179670932277",
* "auth\_uri": "https://accounts.google.com/o/oauth2/auth",
* "token\_uri": "https://oauth2.googleapis.com/token",
* "auth\_provider\_x509\_cert\_url": "https://www.googleapis.com/oauth2/v1/certs",
* "client\_x509\_cert\_url": "https://www.googleapis.com/robot/v1/metadata/x509/bayan-futyan%40my-sheets-api-project-460216.iam.gserviceaccount.com",
* "universe\_domain": "googleapis.com"
* }
* **controllers**
* authController.js
* const db = require('../config/db');
* const bcrypt = require('bcrypt');
* const jwt = require('jsonwebtoken');
* const JWT\_SECRET = 'your\_secret\_key';
* exports.login = (req, res) => {
* const { email, password } = req.body;
* if (!email || !password) {
* return res.status(400).json({ error: 'Email and password are required' });
* }
* const query = 'SELECT \* FROM users WHERE email = ?';
* db.query(query, [email], async (err, results) => {
* if (err) return res.status(500).json({ error: err.message });
* if (results.length === 0) {
* return res.status(401).json({ error: 'Invalid email or password' });
* }
* const user = results[0];
* const match = await bcrypt.compare(password, user.password);
* if (!match) {
* return res.status(401).json({ error: 'Invalid email or password' });
* }
* const token = jwt.sign(
* { user\_id: user.user\_id, role: user.role },
* JWT\_SECRET,
* { expiresIn: '2h' }
* );
* res.status(200).json({
* message: 'Login successful',
* token,
* user: {
* id: user.user\_id,
* name: user.name,
* email: user.email,
* role: user.role
* }
* });
* });
* };
* deliveryTrackingController.js
* // controllers/deliveryTrackingController.js
* const DeliveryTracking = require('../models/deliveryTrackingModel');
* exports.assignDelivery = async (req, res) => {
* try {
* const { deliveryAgentId, taskId } = req.body;
* if (!deliveryAgentId || !taskId) {
* return res.status(400).json({ message: 'Missing deliveryAgentId or taskId' });
* }
* await DeliveryTracking.assignDeliveryAgent(deliveryAgentId, taskId);
* const orphanageUserId = await DeliveryTracking.getOrphanageUserIdByTaskId(taskId);
* if (orphanageUserId) {
* const message = `✅ A delivery agent has accepted to deliver your donation (Logistics Task #${taskId}).`;
* await DeliveryTracking.addNotification(orphanageUserId, message);
* }
* res.status(200).json({ message: 'Task assigned and orphanage notified.' });
* } catch (err) {
* console.error('Assign delivery error:', err);
* res.status(500).json({ message: 'Internal server error.' });
* }
* };
* exports.confirmDelivery = async (req, res) => {
* try {
* const { taskId } = req.body;
* if (!taskId) {
* return res.status(400).json({ message: 'Missing taskId' });
* }
* await DeliveryTracking.confirmDeliveryCompletion(taskId);
* const orphanageUserId = await DeliveryTracking.getOrphanageUserIdByTaskId(taskId);
* if (orphanageUserId) {
* const message = ` Your donation (Task #${taskId}) has been successfully delivered by the delivery agent.`;
* await DeliveryTracking.addNotification(orphanageUserId, message);
* console.log("✅ Notification sent to orphanage:", orphanageUserId);
* } else {
* console.warn("⚠️ No orphanage user found for task:", taskId);
* }
* res.status(200).json({ message: 'Delivery marked as completed, driver set to Active, and orphanage notified.' });
* } catch (err) {
* console.error('Confirm delivery error:', err);
* res.status(500).json({ message: 'Internal server error.' });
* }
* };
* donationController.js
* const Donation = require('../models/donationModel.js');
* exports.createDonation = async (req, res) => {
* try {
* const donationData = req.body;
* const result = await Donation.createDonation(donationData);
* res.status(201).json({
* message: 'Donation created successfully!',
* donationId: result.insertId
* });
* } catch (error) {
* console.error('Error creating donation:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* exports.getAllDonations = async (req, res) => {
* try {
* const donations = await Donation.getAllDonations();
* res.status(200).json(donations);
* } catch (error) {
* console.error('Error fetching donations:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* exports.getAllDonations = async (req, res) => {
* try {
* const { category } = req.query;
* if (category) {
* const donations = await Donation.getDonationsByCategory(category);
* return res.status(200).json(donations);
* }
* const donations = await Donation.getAllDonations();
* res.status(200).json(donations);
* } catch (error) {
* console.error('Error fetching donations:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* exports.getAllDonations = async (req, res) => {
* try {
* const { category, donor\_id } = req.query;
* if (category) {
* const donations = await Donation.getDonationsByCategory(category);
* return res.status(200).json(donations);
* }
* if (donor\_id) {
* const donations = await Donation.getDonationsByDonor(donor\_id);
* return res.status(200).json(donations);
* }
* const donations = await Donation.getAllDonations();
* res.status(200).json(donations);
* } catch (error) {
* console.error('Error fetching donations:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* exports.updateDonationFields = async (req, res) => {
* const donation\_id = req.params.donation\_id;
* const updateFields = req.body;
* try {
* await Donation.updateDonationFields(donation\_id, updateFields);
* if (updateFields.impact\_description && updateFields.impact\_description.trim() !== "") {
* const user\_id = await Donation.getUserIdByDonationId(donation\_id);
* if (user\_id) {
* const message = `Your donation has been disbursed in : ${updateFields.impact\_description} please inter your feedback ^\_^`;
* await Donation.createNotification(user\_id, message);
* }
* }
* res.status(200).json({ message: 'Donation updated successfully.' });
* } catch (err) {
* console.error("Error updating donation:", err);
* res.status(500).json({ message: 'Error updating donation', error: err });
* }
* };
* exports.getDonationsNoImpact = async (req, res) => {
* try {
* const donations = await Donation.getDonationsNoImpact();
* res.status(200).json(donations);
* } catch (error) {
* console.error('Error fetching donations with null impact:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* donationPaymentController.js
* const DonationPayment = require('../models/donationPaymentModel');
* exports.getAllPayments = (req, res) => {
* DonationPayment.getAll((err, results) => {
* if (err) return res.status(500).json({ error: err });
* res.json(results);
* });
* };
* exports.getPaymentById = (req, res) => {
* const id = req.params.id;
* DonationPayment.getById(id, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* if (result.length === 0) return res.status(404).json({ message: 'Payment not found' });
* res.json(result[0]);
* });
* };
* exports.createPayment = (req, res) => {
* DonationPayment.create(req.body, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.status(201).json({ message: 'Payment created', id: result.insertId });
* });
* };
* exports.deletePayment = (req, res) => {
* const id = req.params.id;
* DonationPayment.delete(id, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.json({ message: 'Payment deleted' });
* });
* };
* exports.getSummary = (req, res) => {
* DonationPayment.getSummary((err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.json(result[0]);
* });
* };
* exports.updateTransactionFeePercent = (req, res) => {
* const newPercent = req.body.transaction\_fee\_percent;
* if (typeof newPercent !== 'number' || newPercent < 0 || newPercent > 1) {
* return res.status(400).json({ error: 'Invalid transaction\_fee\_percent. It must be a number between 0 and 1.' });
* }
* DonationPayment.updateTransactionFeePercent(newPercent, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.json({ message: 'Transaction fee percent updated for all records' });
* });
* };
* donationreqController.js
* const LogisticReq = require('../models/donationreqModel');
* const Notification = require('../models/notificationModel');
* exports.getAllLogisticReq = async (req, res) => {
* try {
* const LogisticReqs = await LogisticReq.getAllLogisticReq();
* res.status(200).json(LogisticReqs);
* } catch (error) {
* console.error('Error fetching LogisticReqs:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* exports.createLogisticReq = async (req, res) => {
* try {
* const newReqData = req.body;
* const result = await LogisticReq.createLogisticReq(newReqData);
* if (newReqData.is\_emergency === true || newReqData.is\_emergency === 'true') {
* await Notification.sendEmergencyNotificationsToAllUsers();
* }
* res.status(201).json({
* message: 'Logistic request created successfully',
* id: result.insertId
* });
* } catch (error) {
* console.error('Error creating LogisticReq:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* humanRequestController.js
* const HumanRequest = require('../models/humanRequest.model');
* const MatchModel = require('../models/match.model');
* exports.getAllRequests = (req, res) => {
* HumanRequest.getAllRequests((err, results) => {
* if (err) {
* console.error('Error fetching requests:', err.message);
* return res.status(500).json({ error: 'Failed to retrieve requests' });
* }
* res.status(200).json(results);
* });
* };
* exports.createRequest = (req, res) => {
* const orphanage\_id = req.params.orphanage\_id;
* const { description, is\_emergency, skill\_id } = req.body;
* if (!description) {
* return res.status(400).json({ error: 'Description is required' });
* }
* HumanRequest.insertRequest({ orphanage\_id, description, is\_emergency, skill\_id }, (err, result) => {
* if (err) return res.status(500).json({ error: 'Database error' });
* const request\_id = result.insertId;
* MatchModel.findVolunteersByRequest(request\_id, (err2, volunteers) => {
* if (err2) return res.status(500).json({ message: 'Matching failed' });
* if (volunteers.length === 0) {
* return res.status(201).json({
* message: 'Request created ✅ but no matching volunteers found.',
* request\_id
* });
* }
* const values = volunteers.map(v => [v.volunteer\_id, request\_id]);
* MatchModel.insertVolunteerMatches(values, (err3) => {
* if (err3) return res.status(500).json({ message: 'Matched, but insert failed' });
* const notifyMessage = ` Fortunately, ${volunteers.length} volunteers have been found for your request, go and check. ^\_^`;
* MatchModel.notifyOrphanage(orphanage\_id, notifyMessage, (notifErr) => {
* if (notifErr) {
* return res.status(500).json({ message: 'Matched, but notification failed' });
* }
* res.status(201).json({
* message: 'Request created ✅, volunteers matched, and orphanage notified.',
* request\_id,
* matched\_volunteers: volunteers
* });
* });
* });
* });
* });
* };
* logisticsController.js
* const LogisticsTask = require('../models/logisticsModel');
* exports.getAllTasks = (req, res) => {
* LogisticsTask.getAll((err, results) => {
* if (err) return res.status(500).json({ error: err });
* res.json(results);
* });
* };
* exports.getTaskById = (req, res) => {
* const id = req.params.id;
* LogisticsTask.getById(id, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* if (result.length === 0) return res.status(404).json({ message: 'Task not found' });
* res.json(result[0]);
* });
* };
* exports.createTask = (req, res) => {
* LogisticsTask.create(req.body, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.status(201).json({ message: 'Task created', id: result.insertId });
* });
* };
* exports.updateTask = (req, res) => {
* const id = req.params.id;
* LogisticsTask.update(id, req.body, (err) => {
* if (err) return res.status(500).json({ error: err });
* res.json({ message: 'Task updated' });
* });
* };
* exports.deleteTask = (req, res) => {
* const id = req.params.id;
* LogisticsTask.delete(id, (err) => {
* if (err) return res.status(500).json({ error: err });
* res.json({ message: 'Task deleted' });
* });
* };
* match.controller.js
* const MatchModel = require('../models/match.model');
* const HumanRequest = require('../models/humanRequest.model');
* const notification = require('../models/notificationModel');
* exports.updateRequestStatusAndNotify = (req, res) => {
* const request\_id = req.params.request\_id;
* const { status } = req.body;
* if (!['accepted', 'rejected'].includes(status)) {
* return res.status(400).json({ error: 'Invalid status' });
* }
* HumanRequest.updateRequestStatus(request\_id, status, (err) => {
* if (err) return res.status(500).json({ error: 'Failed to update request status' });
* if (status === 'accepted') {
* MatchModel.findVolunteersByRequest2(request\_id, (err2, volunteers) => {
* if (err2) return res.status(500).json({ error: 'Failed to fetch volunteers' });
* if (volunteers.length === 0) {
* return res.status(200).json({ message: 'There is no request with this ID.' });
* }
* const notifyPromises = volunteers.map(v => {
* const message = `📢 A new help request matches your skills. Please check request #${request\_id}, and if you can help, please accept it.`;
* return new Promise((resolve, reject) => {
* notification.notifyVolunteer(v.volunteer\_id, message, (notifErr) => {
* if (notifErr) reject(notifErr);
* else resolve();
* });
* });
* });
* Promise.all(notifyPromises)
* .then(() => {
* res.status(200).json({ message: 'Request accepted and volunteers notified.' });
* })
* .catch(() => {
* res.status(500).json({ message: 'Request accepted but failed to notify some volunteers.' });
* });
* });
* } else {
* res.status(200).json({ message: 'Request status updated.' });
* }
* });
* };
* exports.getMatchingVolunteers = (req, res) => {
* const request\_id = req.params.request\_id;
* MatchModel.findVolunteersByRequest(request\_id, (err, results) => {
* if (err) {
* console.error('Error fetching matches:', err.message);
* return res.status(500).json({ error: 'Failed to find matching volunteers' });
* }
* if (results.length === 0) {
* return res.status(404).json({ message: 'No matching volunteers found' });
* }
* res.status(200).json(results);
* });
* };
* exports.volunteerRespondToRequest = (req, res) => {
* const {request\_id } = req.params;
* const status = req.body.status || req.body.response;
* if (!['confirmed', 'rejected'].includes(status)) {
* return res.status(400).json({ error: 'Invalid status. Use confirmed or rejected.' });
* }
* const message = status === 'confirmed'
* ? `✅ Volunteer has accepted request #${request\_id}.`
* : `❌ Volunteer has rejected request #${request\_id}.`;
* MatchModel.getOrphanageByRequestId(request\_id, (err, orphanage) => {
* if (err || !orphanage) {
* return res.status(500).json({ error: 'Failed to fetch orphanage information.' });
* }
* const orphanage\_id = orphanage.orphanage\_id;
* notification.notifyOrphanage(orphanage.orphanage\_id, message, (notifErr) => {
* if (notifErr) {
* console.error('Notification error:', notifErr);
* }
* if (status === 'confirmed') {
* HumanRequest.updateRequestStatus(request\_id, 'confirmed', (updateErr) => {
* if (updateErr) {
* return res.status(500).json({ error: 'Failed to confirm request' });
* }
* return res.status(200).json({ message: 'Volunteer accepted and orphanage notified. Status updated to confirmed.' });
* });
* } else {
* return res.status(200).json({ message: 'Volunteer rejected and orphanage notified.' });
* }
* });
* });
* };
* notificationController.js
* const Notification = require('../models/notificationModel');
* exports.getAllNotifications = async (req, res) => {
* try {
* const notifications = await Notification.getAllNotifications();
* res.status(200).json(notifications);
* } catch (error) {
* console.error('Error fetching notifications:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* exports.getNotificationsByDonor = async (req, res) => {
* try {
* const donor\_id = req.params.donor\_id;
* const notifications = await Notification.getNotificationsByDonorId(donor\_id);
* res.status(200).json(notifications);
* } catch (error) {
* console.error('Error fetching notifications for donor:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* exports.sendEmergencyNotificationsToAllUsers = async (req, res) => {
* try {
* const result = await Notification.sendEmergencyNotificationsToAllUsers();
* res.status(200).json({ message: result });
* } catch (error) {
* console.error('Error sending emergency notifications:', error);
* res.status(500).json({ message: 'Failed to send emergency notifications' });
* }
* };
* orphanController.js
* const Orphan = require('../models/orphanModel');
* const db = require('../config/db');
* const util = require('util');
* const query = util.promisify(db.query).bind(db);
* exports.getAllOrphans = async (req, res) => {
* try {
* const orphans = await Orphan.getAll();
* res.json(orphans);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.getOrphanById = async (req, res) => {
* try {
* const orphan = await Orphan.getById(req.params.id);
* if (!orphan) return res.status(404).json({ message: 'Orphan not found' });
* res.json(orphan);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.createOrphan = async (req, res) => {
* try {
* const newOrphan = req.body;
* const insertedId = await Orphan.create(newOrphan);
* const sponsors = await query("SELECT user\_id FROM users WHERE role = 'sponsor'");
* const message = 'A new orphan has been added. If you like to sponsor them';
* for (const sponsor of sponsors) {
* await query(
* 'INSERT INTO notifications (user\_id, message) VALUES (?, ?)',
* [sponsor.user\_id, message]
* );
* }
* res.status(201).json({ id: insertedId, message: 'Orphan created successfully and notifications sent to sponsors.' });
* } catch (err) {
* console.error('Error creating orphan and sending notifications:', err);
* res.status(500).json({ error: err.message });
* }
* };
* exports.updateOrphan = async (req, res) => {
* try {
* await Orphan.update(req.params.id, req.body);
* res.json({ message: 'Orphan updated successfully' });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.softDeleteOrphan = async (req, res) => {
* try {
* const notifications = await Orphan.softDelete(req.params.id);
* res.json({
* message: 'Orphan marked as Not Available',
* notifications: notifications
* });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* orphanUpdateController.js
* const OrphanUpdate = require('../models/orphanUpdateModel');
* exports.getAllUpdates = async (req, res) => {
* try {
* const updates = await OrphanUpdate.getAll();
* res.json(updates);
* } catch (error) {
* console.error("❌ Error in getAllUpdates:", error);
* res.status(500).json({ error: 'خطأ في السيرفر' });
* }
* };
* exports.getUpdatesByOrphan = async (req, res) => {
* try {
* const updates = await OrphanUpdate.getByOrphanId(req.params.id);
* res.json(updates);
* } catch (error) {
* console.error("❌ Error in getUpdatesByOrphan:", error);
* res.status(500).json({ error: 'حدث خطأ أثناء جلب التحديثات' });
* }
* };
* exports.createUpdate = async (req, res) => {
* try {
* const { orphan\_id, update\_type, description } = req.body;
* const image\_url = req.file ? `/images/${req.file.filename}` : null;
* const insertId = await OrphanUpdate.create({
* orphan\_id,
* update\_type,
* description,
* image\_url,
* });
* res.status(201).json({ id: insertId, message: 'تمت إضافة التحديث بنجاح' });
* } catch (error) {
* console.error("❌ Error in createUpdate:", error);
* res.status(500).json({ error: 'فشل في إضافة التحديث' });
* }
* };
* exports.updateUpdate = async (req, res) => {
* try {
* const { update\_type, description } = req.body;
* const image\_url = req.file ? `/images/${req.file.filename}` : null;
* await OrphanUpdate.update(req.params.id, {
* update\_type,
* description,
* image\_url,
* });
* res.json({ message: 'تم تحديث البيانات بنجاح' });
* } catch (error) {
* console.error("❌ Error in updateUpdate:", error);
* res.status(500).json({ error: 'فشل في التحديث' });
* }
* };
* exports.deleteUpdate = async (req, res) => {
* try {
* await OrphanUpdate.delete(req.params.id);
* res.json({ message: 'تم حذف التحديث بنجاح' });
* } catch (error) {
* console.error("❌ Error in deleteUpdate:", error);
* res.status(500).json({ error: 'فشل في الحذف' });
* }
* };
* partnerController.js
* const Partner = require('../models/partnerModel');
* exports.getAllPartners = (req, res) => {
* Partner.getAll((err, results) => {
* if (err) return res.status(500).json({ error: err });
* res.json(results);
* });
* };
* exports.getPartnerById = (req, res) => {
* const id = req.params.id;
* Partner.getById(id, (err, results) => {
* if (err) return res.status(500).json({ error: err });
* if (results.length === 0) return res.status(404).json({ message: 'Partner not found' });
* res.json(results[0]);
* });
* };
* exports.createPartner = (req, res) => {
* Partner.create(req.body, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.status(201).json({ message: 'Partner created', id: result.insertId });
* });
* };
* exports.updatePartner = (req, res) => {
* const id = req.params.id;
* Partner.update(id, req.body, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.json({ message: 'Partner updated' });
* });
* };
* exports.deletePartner = (req, res) => {
* const id = req.params.id;
* Partner.delete(id, (err, result) => {
* if (err) return res.status(500).json({ error: err });
* res.json({ message: 'Partner deleted' });
* });
* };
* skill.controller.js
* const SkillModel = require('../models/skill.model');
* exports.createSkill = (req, res) => {
* const { name } = req.body;
* if (!name) return res.status(400).json({ error: 'Skill name is required' });
* SkillModel.insertSkill(name, (err, result) => {
* if (err) return res.status(500).json({ error: 'Failed to add skill' });
* res.status(201).json({ message: 'Skill added', skill\_id: result.insertId, name });
* });
* };
* exports.getAllSkills = (req, res) => {
* SkillModel.getAllSkills((err, results) => {
* if (err) return res.status(500).json({ error: 'Failed to retrieve skills' });
* res.status(200).json(results);
* });
* };
* exports.addSkills = (req, res) => {
* const volunteer\_id = req.params.volunteer\_id;
* const { skill\_ids, availability } = req.body;
* if (!Array.isArray(skill\_ids) || skill\_ids.length === 0) {
* return res.status(400).json({ error: 'skill\_ids must be a non-empty array' });
* }
* const finish = () => {
* SkillModel.addSkillsToVolunteer(volunteer\_id, skill\_ids, (err2, result) => {
* if (err2) {
* console.error('Error adding skills:', err2.message);
* return res.status(500).json({ error: 'Failed to assign skills' });
* }
* res.status(201).json({
* message: 'Skills and availability saved successfully',
* addedCount: result.affectedRows
* });
* });
* };
* if (availability) {
* SkillModel.updateAvailability(volunteer\_id, availability, (err) => {
* if (err) {
* console.error('Error updating availability:', err.message);
* return res.status(500).json({ error: 'Failed to update availability' });
* }
* finish();
* });
* } else {
* finish();
* }
* };
* sponsorController.js
* const Sponsor = require('../models/sponsorModel');
* const { exportSponsors } = require('../Apis/exportSponsorsToSheet');
* const axios = require('axios');
* exports.getAllSponsors = async (req, res) => {
* try {
* const sponsors = await Sponsor.getAll();
* res.json(sponsors);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.getSponsorById = async (req, res) => {
* try {
* const sponsor = await Sponsor.getById(req.params.id);
* if (!sponsor) return res.status(404).json({ message: 'Sponsor not found' });
* res.json(sponsor);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.updateSponsor = async (req, res) => {
* try {
* await Sponsor.update(req.params.id, req.body);
* res.json({ message: 'Sponsor updated successfully' });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.exportSponsorsToGoogleSheet = async (req, res) => {
* try {
* const result = await exportSponsors();
* if (result.success) {
* res.status(200).json({ message: "Sponsors exported successfully", count: result.count });
* } else {
* res.status(500).json({ message: "Export failed", error: result.error });
* }
* } catch (err) {
* res.status(500).json({ message: "Export failed", error: err.message });
* }
* };
* exports.postSponsorByIdToExternalAPI = async (req, res) => {
* try {
* const sponsor = await Sponsor.getById(req.params.id);
* if (!sponsor) {
* return res.status(404).json({ message: 'Sponsor not found' });
* }
* const externalApiUrl = 'https://script.google.com/macros/s/AKfycbyabSlYrD9fHhDMXUiAmoQTHOe72mgzdbMN2Ni0veodS8knX4FD5VEV9to1D7EhPke3uQ/exec';
* const response = await axios.post(externalApiUrl, sponsor);
* res.status(200).json({
* message: 'Sponsor data posted successfully to external API',
* externalResponse: response.data
* });
* } catch (err) {
* res.status(500).json({ message: 'Failed to post sponsor to external API', error: err.message });
* }
* };
* sponsorshipController.js
* const Sponsorship = require('../models/sponsorshipModel');
* const db = require('../config/db');
* exports.getAllSponsorships = async (req, res) => {
* try {
* const results = await Sponsorship.getAll();
* res.json(results);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.getByOrphanId = async (req, res) => {
* try {
* const results = await Sponsorship.getByOrphanId(req.params.orphanId);
* res.json(results);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.getBySponsorId = async (req, res) => {
* try {
* const results = await Sponsorship.getBySponsorId(req.params.sponsorId);
* res.json(results);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.createSponsorship = async (req, res) => {
* try {
* const insertedId = await Sponsorship.create(req.body);
* res.status(201).json({ id: insertedId, message: 'Your request is pending approval' });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.updateSponsorship = async (req, res) => {
* try {
* await Sponsorship.update(req.params.id, req.body);
* res.json({ message: 'Sponsorship updated successfully' });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* function queryPromise(sql, params) {
* return new Promise((resolve, reject) => {
* db.query(sql, params, (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* }
* exports.updateSponsorshipStatus = async (req, res) => {
* try {
* const { status } = req.body;
* const id = req.params.id;
* // تحديث الحالة
* await Sponsorship.updateStatus(id, status);
* // جلب بيانات ال sponsorship مع اسم ال orphan و user\_id للسبونسر
* const query = `
* SELECT s.sponsor\_id, o.name AS orphan\_name, u.user\_id
* FROM sponsorships s
* JOIN orphans o ON s.orphan\_id = o.id
* JOIN sponsors sp ON s.sponsor\_id = sp.id
* JOIN users u ON sp.user\_id = u.user\_id
* WHERE s.id = ?
* `;
* const rows = await queryPromise(query, [id]);
* if (rows.length === 0) {
* return res.status(404).json({ error: 'Sponsorship not found' });
* }
* const sponsorUserId = rows[0].user\_id;
* const orphanName = rows[0].orphan\_name;
* // رسالة النوتيفيكيشن
* let message = '';
* if (status === 'Available') {
* message = `Your sponsorship request for ${orphanName} has been approved.`;
* } else if (status === 'Pending Approval') {
* message = `Your sponsorship request for ${orphanName} is pending approval.`;
* } else if (status === 'Rejected') {
* message = `Your sponsorship request for ${orphanName} has been rejected.`;
* } else {
* message = `Status of your sponsorship request for ${orphanName} has been updated to ${status}.`;
* }
* const insertNotificationQuery = `
* INSERT INTO notifications (user\_id, message, created\_at)
* VALUES (?, ?, NOW())
* `;
* await queryPromise(insertNotificationQuery, [sponsorUserId, message]);
* res.json({ message: 'Sponsorship status updated and notification sent successfully' });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.softDeleteSponsorship = async (req, res) => {
* try {
* await Sponsorship.softDelete(req.params.id);
* res.json({ message: 'Sponsorship marked as Not Available' });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* userController.js
* const User = require('../models/userModel');
* const bcrypt = require('bcrypt');
* exports.createUser = async (req, res) => {
* try {
* const { name, email, password, phone, location, role, status } = req.body;
* if (!['sponsor', 'donor', 'volunteer', 'delivery', 'orphanages'].includes(role)) {
* return res.status(400).json({ error: 'Invalid role' });
* }
* const hashedPassword = await bcrypt.hash(password, 10);
* const newUser = {
* name,
* email,
* password: hashedPassword,
* phone,
* location,
* role,
* status: status || 'active',
* created\_at: new Date()
* };
* const userId = await User.create(newUser);
* await User.insertToRoleTable(role, userId);
* res.status(201).json({ message: 'User created successfully', userId });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.getAllUsers = async (req, res) => {
* try {
* const users = await User.getAllUsers();
* res.status(200).json(users);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.getUserById = async (req, res) => {
* try {
* const userId = req.params.id;
* const user = await User.getUserById(userId);
* if (!user) {
* return res.status(404).json({ error: 'User not found' });
* }
* res.status(200).json(user);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.getUsersByRole = async (req, res) => {
* try {
* const role = req.params.role.toLowerCase();
* if (!['sponsor', 'donor', 'volunteer', 'delivery', 'orphanages'].includes(role)) {
* return res.status(400).json({ error: 'Invalid role' });
* }
* const users = await User.getUsersByRole(role);
* res.status(200).json(users);
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.updateUser = async (req, res) => {
* try {
* const userId = req.params.id;
* const { name, phone, location, status } = req.body;
* const updatedFields = { name, phone, location, status };
* // إزالة الحقول undefined (يعني مش مرسلة)
* Object.keys(updatedFields).forEach(key => {
* if (updatedFields[key] === undefined) {
* delete updatedFields[key];
* }
* });
* if (Object.keys(updatedFields).length === 0) {
* return res.status(400).json({ error: 'No fields provided for update' });
* }
* await User.update(userId, updatedFields);
* res.status(200).json({ message: 'User updated successfully' });
* } catch (err) {
* res.status(500).json({ error: err.message });
* }
* };
* exports.softDeleteUser = async (req, res) => {
* try {
* const userId = req.params.id;
* const user = await User.getUserById(userId);
* if (!user) {
* return res.status(404).json({ message: 'User not found' });
* }
* await User.softDelete(userId);
* res.status(200).json({ message: 'User status updated to suspended' });
* } catch (err) {
* res.status(500).json({ message: 'Error suspending user', error: err.message });
* }
* };
* volunteerController.jsimages
* const volunteer = require('../models/volunteerModel.js');
* exports.getAllVolunteers = async (req, res) => {
* try {
* const Volunteers = await volunteer.getAllVolunteers();
* res.status(200).json(Volunteers);
* } catch (error) {
* console.error('Error fetching Volunteers:', error);
* res.status(500).json({ message: 'Internal server error' });
* }
* };
* **middlewares**

**checkPermission.js**

* module.exports = (requiredPermission) => {
* return (req, res, next) => {
* const user = req.user;
* if (!user || !user.role) {
* return res.status(403).json({ message: 'Access denied. User role missing.' });
* }
* const rolePermissions = {
* admin: ['canViewSponsors', 'canEditSponsors', 'canExportSponsors', 'canManageUsers', 'canApproveSponsorships'],
* sponsor: ['canViewOwnSponsorships','canRequestSponsorship','canEditOwnProfile'],
* donor: ['canViewOwnDonations','canMakeDonation','canEditOwnProfile'],
* volunteer: ['canViewAssignedTasks','canMarkTaskComplete','canEditOwnProfile'],
* orphanages: ['canViewOrphans','canAddOrphan', 'canUpdateOrphan','canSendOrphanUpdates'],
* delivery: ['canViewDeliveryTasks','canUpdateDeliveryStatus','canViewAssignedDeliveries']
* };
* const permissions = rolePermissions[user.role] || [];
* if (!permissions.includes(requiredPermission)) {
* return res.status(403).json({ message: 'Access denied. Permission not granted.' });
* }
* next();
* };
* };

**verifyToken.js**

* const jwt = require('jsonwebtoken');
* module.exports = (req, res, next) => {
* const authHeader = req.headers.authorization;
* if (!authHeader || !authHeader.startsWith('Bearer ')) {
* return res.status(401).json({ message: 'Access denied. No token provided.' });
* }
* const token = authHeader.split(' ')[1];
* try {
* const decoded = jwt.verify(token, process.env.JWT\_SECRET || 'your\_secret\_key');
* req.user = decoded;
* next();
* } catch (err) {
* res.status(400).json({ message: 'Invalid token.' });
* }
* };
* **models**
* **deliveryTrackingModel.js**
* const connection = require('../config/db');
* // ربط المهمة بالسائق وتحديث حالته
* exports.assignDeliveryAgent = (deliveryAgentId, taskId) => {
* return new Promise((resolve, reject) => {
* connection.query(
* `UPDATE logistics\_tasks SET delivery\_id = ?, status = 'in\_progress' WHERE id = ?`,
* [deliveryAgentId, taskId],
* (err) => {
* if (err) return reject(err);
* connection.query(
* `UPDATE delivery SET status = 'Busy' WHERE id = ?`,
* [deliveryAgentId],
* (err2) => {
* if (err2) return reject(err2);
* resolve();
* }
* );
* }
* );
* });
* };
* // جلب user\_id للمؤسسة المرتبطة بالتبرع
* exports.getOrphanageUserIdByTaskId = (taskId) => {
* return new Promise((resolve, reject) => {
* connection.query(`
* SELECT u.user\_id
* FROM logistics\_tasks lt
* JOIN donations d ON lt.donation\_id = d.donation\_id
* JOIN orphanages o ON d.orphanage\_id = o.orphanage\_id
* JOIN users u ON o.user\_id = u.user\_id
* WHERE lt.id = ?
* `, [taskId], (err, results) => {
* if (err) return reject(err);
* resolve(results.length ? results[0].user\_id : null);
* });
* });
* };
* // إضافة إشعار
* exports.addNotification = (userId, message) => {
* return new Promise((resolve, reject) => {
* connection.query(
* `INSERT INTO notifications (user\_id, message) VALUES (?, ?)`,
* [userId, message],
* (err, result) => {
* if (err) {
* console.error("❌ Error adding notification:", err); // أضف هذا
* return reject(err);
* }
* console.log("✅ Notification added for user:", userId); // أضف هذا
* resolve(result);
* }
* );
* });
* };
* // تأكيد وصول المهمة وتحديث الحالات
* exports.confirmDeliveryCompletion = (taskId) => {
* return new Promise((resolve, reject) => {
* // أولاً نجيب ID السائق من المهمة
* connection.query(
* `SELECT delivery\_id FROM logistics\_tasks WHERE id = ?`,
* [taskId],
* (err, results) => {
* if (err) return reject(err);
* if (!results.length) return reject(new Error("Task not found"));
* const deliveryAgentId = results[0].delivery\_id;
* // نحدّث حالة المهمة والسائق معًا
* connection.query(
* `UPDATE logistics\_tasks SET status = 'completed' WHERE id = ?`,
* [taskId],
* (err2) => {
* if (err2) return reject(err2);
* connection.query(
* `UPDATE delivery SET status = 'Active' WHERE id = ?`,
* [deliveryAgentId],
* (err3) => {
* if (err3) return reject(err3);
* resolve();
* }
* );
* }
* );
* }
* );
* });
* };
* **donationModel.js**
* // models/donationModel.js
* const db = require('../config/db');
* exports.createDonation = (data) => {
* return new Promise((resolve, reject) => {
* const sql = `
* INSERT INTO donations (
* donor\_id, orphanage\_id,
* category\_name, category\_description,
* is\_emergency
* ) VALUES (?, ?, ?, ?, ?)
* `
* const values = [
* data.donor\_id,
* data.orphanage\_id,
* data.amount,
* data.category\_name,
* data.category\_description,
* data.is\_emergency || 'False'
* ];
* db.query(sql, values, (err, result) => {
* if (err) return reject(err);
* resolve(result);
* });
* });
* };
* exports.getAllDonations = () => {
* return new Promise((resolve, reject) => {
* const sql = `SELECT \* FROM donations ORDER BY donation\_date DESC`;
* db.query(sql, (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* exports.getDonationsByCategory = (category\_name) => {
* return new Promise((resolve, reject) => {
* const sql = `
* SELECT
* d.\*,
* u.name AS donor\_name,
* o.name AS orphanage\_name
* FROM donations d
* JOIN donors dn ON d.donor\_id = dn.donor\_id
* JOIN users u ON dn.user\_id = u.user\_id
* JOIN orphanages o ON d.orphanage\_id = o.orphanage\_id
* WHERE d.category\_name = ?
* ORDER BY d.donation\_date DESC
* `;
* db.query(sql, [category\_name], (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* exports.getDonationsByDonor = (donor\_id) => {
* return new Promise((resolve, reject) => {
* const sql = `
* SELECT
* d.\*,
* u.name AS donor\_name,
* o.name AS orphanage\_name
* FROM donations d
* JOIN donors dn ON d.donor\_id = dn.donor\_id
* JOIN users u ON dn.user\_id = u.user\_id
* JOIN orphanages o ON d.orphanage\_id = o.orphanage\_id
* WHERE d.donor\_id = ?
* ORDER BY d.donation\_date DESC
* `;
* db.query(sql, [donor\_id], (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* exports.getDonationsNoImpact = () => {
* return new Promise((resolve, reject) => {
* const sql = `SELECT \* FROM donations WHERE impact\_description IS NULL ORDER BY donation\_date DESC`;
* db.query(sql, (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* // تحديث مرن لأي حقل
* exports.updateDonationFields = (donation\_id, fields) => {
* return new Promise((resolve, reject) => {
* const keys = Object.keys(fields);
* const values = Object.values(fields);
* if (keys.length === 0) {
* return reject(new Error("No fields to update"));
* }
* const sql = `
* UPDATE donations
* SET ${keys.map(key => `${key} = ?`).join(", ")}
* WHERE donation\_id = ?
* `;
* db.query(sql, [...values, donation\_id], (err, result) => {
* if (err) return reject(err);
* resolve(result);
* });
* });
* };
* // Get user\_id by donation\_id
* exports.getUserIdByDonationId = (donation\_id) => {
* return new Promise((resolve, reject) => {
* const sql = `
* SELECT u.user\_id
* FROM donations d
* JOIN donors dn ON d.donor\_id = dn.donor\_id
* JOIN users u ON dn.user\_id = u.user\_id
* WHERE d.donation\_id = ?
* `;
* db.query(sql, [donation\_id], (err, results) => {
* if (err) return reject(err);
* if (results.length > 0) {
* resolve(results[0].user\_id);
* } else {
* resolve(null); // donation not found
* }
* });
* });
* };
* // Insert notification
* exports.createNotification = (user\_id, message) => {
* return new Promise((resolve, reject) => {
* const sql = `
* INSERT INTO notifications (user\_id, message)
* VALUES (?, ?)
* `;
* db.query(sql, [user\_id, message], (err, result) => {
* if (err) return reject(err);
* resolve(result);
* });
* });
* };
* **donationPaymentModel.js**
* const db = require('../config/db');
* exports.getAll = (callback) => {
* db.query('SELECT \* FROM donation\_payments', callback);
* };
* exports.getById = (id, callback) => {
* db.query('SELECT \* FROM donation\_payments WHERE id = ?', [id], callback);
* };
* exports.create = (data, callback) => {
* // نأخذ النسبة من data أو نعطيها قيمة افتراضية 2% مثلاً
* const feePercent = data.transaction\_fee\_percent ?? 0.02;
* const transaction\_fee = data.paid\_amount \* feePercent;
* const net\_amount = data.paid\_amount - transaction\_fee;
* const query = `
* INSERT INTO donation\_payments
* (donation\_id, payment\_method, transaction\_id, status, paid\_amount, transaction\_fee, net\_amount, paid\_at, transaction\_fee\_percent)
* VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
* `;
* const params = [
* data.donation\_id,
* data.payment\_method,
* data.transaction\_id,
* data.status,
* data.paid\_amount,
* transaction\_fee,
* net\_amount,
* data.paid\_at,
* feePercent
* ];
* db.query(query, params, callback);
* };
* exports.delete = (id, callback) => {
* db.query('DELETE FROM donation\_payments WHERE id = ?', [id], callback);
* };
* exports.getSummary = (callback) => {
* const query = `
* SELECT
* COUNT(\*) AS total\_transactions,
* SUM(paid\_amount) AS total\_paid\_amount,
* SUM(transaction\_fee) AS total\_transaction\_fees,
* SUM(net\_amount) AS total\_net\_amount
* FROM donation\_payments
* `;
* db.query(query, callback);
* };
* exports.updateTransactionFeePercent = (newPercent, callback) => {
* const query = `
* UPDATE donation\_payments
* SET
* transaction\_fee\_percent = ?,
* transaction\_fee = paid\_amount \* ?,
* net\_amount = paid\_amount - (paid\_amount \* ?)
* `;
* const params = [newPercent, newPercent, newPercent];
* db.query(query, params, callback);
* };
* **donationreqModel.js**
* const db = require('../config/db');
* exports.getAllLogisticReq = () => {
* return new Promise((resolve, reject) => {
* const sql = `SELECT \* FROM donation\_requests ORDER BY requested\_at DESC`;
* db.query(sql, (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* exports.createLogisticReq = (data) => {
* return new Promise((resolve, reject) => {
* const sql = `INSERT INTO donation\_requests
* (orphanage\_id, request\_type, description, status, requested\_at, is\_emergency)
* VALUES (?, ?, ?, ?, ?, ? )`;
* const values = [
* data.orphanage\_id,
* data.request\_type,
* data.description || null,
* data.delivery\_location,
* data.status || 'pending',
* data.requested\_at || new Date(),
* data.delivered\_at || null,
* data.is\_emergency || false
* ];
* db.query(sql, values, (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* **humanRequest.model.js**
* const db = require('../config/db');
* exports.insertRequest = ({ orphanage\_id, description, is\_emergency = 'False', skill\_id }, callback) => {
* const sql = `
* INSERT INTO human\_requests (orphanage\_id, description, is\_emergency, skill\_id, date\_requested)
* VALUES (?, ?, ?, ?, NOW())
* `;
* db.query(sql, [orphanage\_id, description, is\_emergency, skill\_id], callback);
* };
* exports.getAllRequests = (callback) => {
* const sql = `
* SELECT hr.\*, s.name AS skill\_name
* FROM human\_requests hr
* LEFT JOIN skills s ON hr.skill\_id = s.skill\_id
* ORDER BY hr.date\_requested DESC
* `;
* db.query(sql, callback);
* };
* exports.updateRequestStatus = (request\_id, status, callback) => {
* const sql = `UPDATE volunteer\_requests SET status = ? WHERE request\_id = ?`;
* db.query(sql, [status, request\_id], callback);
* };
* **logisticsModel.js**
* const db = require('../config/db');
* exports.getAll = (callback) => {
* db.query('SELECT \* FROM logistics\_tasks', callback);
* };
* exports.getById = (id, callback) => {
* db.query('SELECT \* FROM logistics\_tasks WHERE id = ?', [id], callback);
* };
* exports.create = (data, callback) => {
* const query = `
* INSERT INTO logistics\_tasks
* (donation\_id, delivery\_id, origin\_address, destination\_address, title, description, scheduled\_at, status)
* VALUES (?, ?, ?, ?, ?, ?, ?, ?)
* `;
* const params = [
* data.donation\_id,
* data.delivery\_id,
* data.origin\_address,
* data.destination\_address,
* data.title,
* data.description,
* data.scheduled\_at,
* data.status || 'pending',
* ];
* db.query(query, params, callback);
* };
* exports.update = (id, data, callback) => {
* const query = `
* UPDATE logistics\_tasks SET
* donation\_id = ?,
* delivery\_id = ?,
* origin\_address = ?,
* destination\_address = ?,
* title = ?,
* description = ?,
* scheduled\_at = ?,
* status = ?
* WHERE id = ?
* `;
* const params = [
* data.donation\_id,
* data.delivery\_id,
* data.origin\_address,
* data.destination\_address,
* data.title,
* data.description,
* data.scheduled\_at,
* data.status,
* id,
* ];
* db.query(query, params, callback);
* };
* exports.delete = (id, callback) => {
* db.query('DELETE FROM logistics\_tasks WHERE id = ?', [id], callback);
* };
* **match.model.js**
* const db = require('../config/db');
* exports.findVolunteersByRequest = (request\_id, callback) => {
* const sql = `
* SELECT v.volunteer\_id, u.name, u.email, v.availability, s.name AS skill\_name
* FROM human\_requests hr
* JOIN volunteer\_skills vs ON hr.skill\_id = vs.skill\_id
* JOIN volunteers v ON vs.volunteer\_id = v.volunteer\_id
* JOIN users u ON v.user\_id = u.user\_id
* JOIN skills s ON vs.skill\_id = s.skill\_id
* WHERE hr.request\_id = ?
* `;
* db.query(sql, [request\_id], callback);
* };
* exports.insertVolunteerMatches = (volunteerMatches, callback) => {
* const sql = `INSERT INTO volunteer\_requests (volunteer\_id, request\_id) VALUES ?`;
* db.query(sql, [volunteerMatches], callback);
* };
* exports.notifyOrphanage = (orphanage\_id, message, callback) => {
* const sql = `
* INSERT INTO notifications (user\_id, message)
* SELECT u.user\_id, ? FROM users u
* JOIN orphanages o ON o.user\_id = u.user\_id
* WHERE o.orphanage\_id = ?
* `;
* db.query(sql, [message, orphanage\_id], callback);
* };
* exports.findVolunteersByRequest2 = (request\_id, callback) => {
* const sql = `
* SELECT volunteer\_id
* FROM volunteer\_requests
* WHERE id = ?
* `;
* db.query(sql, [request\_id], callback);
* };
* exports.getOrphanageByRequestId = (request\_id, callback) => {
* const sql = `
* SELECT orphanage\_id
* FROM human\_requests
* WHERE request\_id = (
* SELECT request\_id
* FROM volunteer\_requests
* WHERE id = ?
* )
* `;
* db.query(sql, [request\_id], (err, results) => {
* if (err) return callback(err);
* if (results.length === 0) return callback(null, null);
* const orphanage\_id = results[0].orphanage\_id;
* callback(null, { orphanage\_id });
* });
* };
* **notificationModel.js**
* const db = require('../config/db');
* exports.getAllNotifications = () => {
* return new Promise((resolve, reject) => {
* const sql = `SELECT \* FROM notifications ORDER BY created\_at DESC`;
* db.query(sql, (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* exports.getNotificationsByDonorId = (donor\_id) => {
* return new Promise((resolve, reject) => {
* const sql = `
* SELECT n.\*
* FROM notifications n
* JOIN users u ON n.user\_id = u.user\_id
* JOIN donors d ON d.user\_id = u.user\_id
* WHERE d.donor\_id = ?
* ORDER BY n.created\_at DESC
* `;
* db.query(sql, [donor\_id], (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* };
* //exports.sendEmergencyNotificationsToAllUsers = () => {
* //return new Promise((resolve, reject) => {
* // const message = 'Emergency: A new urgent campaign needs your support ,Plese help as if you can ^\_^ ';
* // const sql = `
* // INSERT INTO notifications (user\_id, message)
* // SELECT user\_id, ? FROM users
* // `;
* // db.query(sql, [message], (err, result) => {
* // if (err) return reject(err);
* // resolve(result);
* // });
* // });
* //};
* const nodemailer = require("nodemailer");
* exports.sendEmergencyNotificationsToAllUsers = () => {
* return new Promise((resolve, reject) => {
* const message = '🚨 Emergency: A new urgent campaign needs your support! Please help if you can. 🙏';
* // 1. نجيب كل المستخدمين اللي عندهم إيميل
* const getUsersSql = `SELECT user\_id, email, name FROM users WHERE status = 'Active'`;
* db.query(getUsersSql, async (err, users) => {
* if (err) return reject(err);
* if (users.length === 0) {
* return resolve("🚫 لا يوجد مستخدمين لإرسال الإيميلات");
* }
* // 2. نحفظ الإشعارات في قاعدة البيانات
* const insertNotificationsSql = `
* INSERT INTO notifications (user\_id, message)
* VALUES ?
* `;
* const notificationValues = users.map(user => [user.user\_id, message]);
* db.query(insertNotificationsSql, [notificationValues], async (err) => {
* if (err) return reject(err);
* // 3. نجهز الإرسال عبر Gmail
* let transporter = nodemailer.createTransport({
* service: 'gmail',
* auth: {
* user: 's12112220@stu.najah.edu',
* pass: 'oenlvorjkkukhcbj' // App Password (بدون فراغات)
* }
* });
* // 4. نرسل الإيميل لكل مستخدم
* for (let user of users) {
* const mailOptions = {
* from: '"Relief System" <your-email@gmail.com>',
* to: user.email,
* subject: '🚨 Urgent Campaign Needs Your Help',
* html: `
* <h2>🚨 حملة طارئة</h2>
* <p>مرحبًا ${user.name || ''}،</p>
* <p>هناك حملة طارئة الآن وتحتاج دعمك! تبرع الآن وساهم في إنقاذ الأرواح 💚</p>
* <a href="https://yourwebsite.com/emergency" style="padding: 10px 20px; background: red; color: white; text-decoration: none;">ساهم الآن</a>
* <p>شكرًا لك 🙏</p>
* `
* };
* try {
* await transporter.sendMail(mailOptions);
* console.log(`✅ تم إرسال الإيميل إلى: ${user.email}`);
* } catch (emailErr) {
* console.error(`❌ فشل في إرسال الإيميل إلى ${user.email}`, emailErr);
* }
* }
* resolve("📨 تم إرسال كل الإشعارات والإيميلات بنجاح!");
* });
* });
* });
* };
* exports.notifyVolunteer = (volunteer\_id, message, callback) => {
* const sql = `
* INSERT INTO notifications (user\_id, message)
* SELECT u.user\_id, ? FROM users u
* WHERE u.user\_id = (
* SELECT user\_id FROM volunteers WHERE volunteer\_id = ?
* )
* `;
* db.query(sql, [message, volunteer\_id], callback);
* };
* exports.notifyVolunteersByRequest = (request\_id, message, callback) => {
* const sql = `
* INSERT INTO notifications (user\_id, message)
* SELECT v.user\_id, ?
* FROM volunteer\_requests vr
* JOIN volunteers v ON vr.volunteer\_id = v.volunteer\_id
* WHERE vr.request\_id = ?
* `;
* db.query(sql, [message, request\_id], callback);
* };
* exports.notifyOrphanage = (orphanage\_id, message, callback) => {
* const sql = `
* INSERT INTO notifications (user\_id, message)
* SELECT u.user\_id, ? FROM users u
* JOIN orphanages o ON o.user\_id = u.user\_id
* WHERE o.orphanage\_id = ?
* `;
* db.query(sql, [message, orphanage\_id], callback);
* };
* **orphanModel.js**
* const db = require('../config/db');
* const Orphan = {
* getAll: () => {
* return new Promise((resolve, reject) => {
* const query = `
* SELECT o.\*, u.name AS orphanage\_name
* FROM orphans o
* JOIN orphanages og ON o.orphanage\_id = og.orphanage\_id
* JOIN users u ON og.user\_id = u.user\_id
* `;
* db.query(query, (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* },
* getById: (id) => {
* return new Promise((resolve, reject) => {
* const query = `
* SELECT o.\*, u.name AS orphanage\_name
* FROM orphans o
* JOIN orphanages og ON o.orphanage\_id = og.orphanage\_id
* JOIN users u ON og.user\_id = u.user\_id
* WHERE o.id = ?
* `;
* db.query(query, [id], (err, results) => {
* if (err) reject(err);
* else resolve(results[0]);
* });
* });
* },
* create: (data) => {
* return new Promise((resolve, reject) => {
* db.query('INSERT INTO orphans SET ?', data, (err, result) => {
* if (err) reject(err);
* else resolve(result.insertId);
* });
* });
* },
* update: (id, data) => {
* return new Promise((resolve, reject) => {
* db.query('UPDATE orphans SET ? WHERE id = ?', [data, id], (err, result) => {
* if (err) return reject(err);
* const updateQuery = `
* INSERT INTO orphan\_updates (orphan\_id, update\_type, description, created\_at)
* VALUES (?, ?, ?, NOW())
* `;
* const updateType = 'profile\_update';
* const description = `Orphan profile updated: ${Object.keys(data).join(', ')}`;
* db.query(updateQuery, [id, updateType, description], (err2) => {
* if (err2) return reject(err2);
* resolve(result);
* });
* });
* });
* },
* softDelete: (id) => {
* return new Promise((resolve, reject) => {
* const orphanStatusQuery = 'SELECT status FROM orphans WHERE id = ?';
* // ✅ استعلام جديد يجلب user\_id بدلاً من sponsor\_id
* const sponsorshipQuery = `
* SELECT DISTINCT s.user\_id
* FROM sponsorships sp
* JOIN sponsors s ON sp.sponsor\_id = s.id
* WHERE sp.orphan\_id = ?
* `;
* const updateSponsorshipStatusQuery = 'UPDATE sponsorships SET status = "Not Available" WHERE orphan\_id = ?';
* const insertUpdateQuery = `
* INSERT INTO orphan\_updates (orphan\_id, update\_type, description)
* VALUES (?, 'status\_change', 'This orphan is no longer available.')
* `;
* const updateOrphanStatusQuery = 'UPDATE orphans SET status = "Not Available" WHERE id = ?';
* const insertNotificationsQuery = 'INSERT INTO notifications (user\_id, message, created\_at) VALUES ?';
* const getOrphanNameQuery = 'SELECT name FROM orphans WHERE id = ?';
* db.query(orphanStatusQuery, [id], (err, results) => {
* if (err) return reject(err);
* if (results.length === 0) {
* return reject(new Error('Orphan not found'));
* }
* if (results[0].status === 'Not Available') {
* return reject(new Error('This orphan is already marked as Not Available.'));
* }
* db.query(getOrphanNameQuery, [id], (errName, nameResults) => {
* if (errName) return reject(errName);
* const orphanName = nameResults[0]?.name || `ID: ${id}`;
* db.query(sponsorshipQuery, [id], (err, sponsorshipResults) => {
* if (err) return reject(err);
* const uniqueUserIds = [...new Set(sponsorshipResults.map(row => row.user\_id))];
* const notifications = [];
* const notificationValues = [];
* if (uniqueUserIds.length > 0) {
* const message = `The orphan (${orphanName}) you were sponsoring is no longer available.`;
* const createdAt = new Date();
* for (const userId of uniqueUserIds) {
* notifications.push(`Notification sent to sponsor with user\_id = ${userId}`);
* notificationValues.push([userId, message, createdAt]);
* }
* }
* db.query(updateOrphanStatusQuery, [id], (err0) => {
* if (err0) return reject(err0);
* db.query(updateSponsorshipStatusQuery, [id], (err2) => {
* if (err2) return reject(err2);
* db.query(insertUpdateQuery, [id], (err3) => {
* if (err3) return reject(err3);
* if (notificationValues.length > 0) {
* db.query(insertNotificationsQuery, [notificationValues], (err4) => {
* if (err4) return reject(err4);
* return resolve(notifications);
* });
* } else {
* return resolve(['No sponsors to notify.']);
* }
* });
* });
* });
* });
* });
* });
* });
* }
* };
* module.exports = Orphan;
* **orphanUpdateModel.js**
* const db = require('../config/db');
* const OrphanUpdate = {
* getAll: () => {
* return new Promise((resolve, reject) => {
* db.query('SELECT \* FROM orphan\_updates', (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* },
* getByOrphanId: (orphanId) => {
* return new Promise((resolve, reject) => {
* db.query('SELECT \* FROM orphan\_updates WHERE orphan\_id = ?', [orphanId], (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* },
* create: (data) => {
* return new Promise((resolve, reject) => {
* const query = 'INSERT INTO orphan\_updates (orphan\_id, update\_type, description, image\_url) VALUES (?, ?, ?, ?)';
* db.query(query, [data.orphan\_id, data.update\_type, data.description, data.image\_url], (err, result) => {
* if (err) reject(err);
* else resolve(result.insertId);
* });
* });
* },
* update: (id, data) => {
* return new Promise((resolve, reject) => {
* const query = 'UPDATE orphan\_updates SET update\_type = ?, description = ?, image\_url = ? WHERE id = ?';
* db.query(query, [data.update\_type, data.description, data.image\_url, id], (err, result) => {
* if (err) reject(err);
* else resolve(result);
* });
* });
* },
* delete: (id) => {
* return new Promise((resolve, reject) => {
* db.query('DELETE FROM orphan\_updates WHERE id = ?', [id], (err, result) => {
* if (err) reject(err);
* else resolve(result);
* });
* });
* }
* };
* module.exports = OrphanUpdate;
* **partnerModel.js**
* const db = require('../config/db');
* exports.getAll = (callback) => {
* db.query('SELECT \* FROM partners', callback);
* };
* exports.getById = (id, callback) => {
* db.query('SELECT \* FROM partners WHERE id = ?', [id], callback);
* };
* exports.create = (data, callback) => {
* const query = `
* INSERT INTO partners (name, contact\_email, phone\_number, partnership\_type, description, created\_at, updated\_at)
* VALUES (?, ?, ?, ?, ?, NOW(), NOW())
* `;
* const params = [
* data.name,
* data.contact\_email,
* data.phone\_number,
* data.partnership\_type,
* data.description
* ];
* db.query(query, params, callback);
* };
* exports.update = (id, data, callback) => {
* const query = `
* UPDATE partners
* SET name = ?, contact\_email = ?, phone\_number = ?, partnership\_type = ?, description = ?, updated\_at = NOW()
* WHERE id = ?
* `;
* const params = [
* data.name,
* data.contact\_email,
* data.phone\_number,
* data.partnership\_type,
* data.description,
* id
* ];
* db.query(query, params, callback);
* };
* exports.delete = (id, callback) => {
* db.query('DELETE FROM partners WHERE id = ?', [id], callback);
* };
* **skill.model.js**
* const db = require('../config/db');
* exports.insertSkill = (name, callback) => {
* const sql = 'INSERT INTO skills (name) VALUES (?)';
* db.query(sql, [name], callback);
* };
* exports.getAllSkills = (callback) => {
* const sql = 'SELECT \* FROM skills';
* db.query(sql, callback);
* };
* exports.updateAvailability = (volunteer\_id, availability, callback) => {
* const sql = 'UPDATE volunteers SET availability = ? WHERE volunteer\_id = ?';
* db.query(sql, [availability, volunteer\_id], callback);
* };
* exports.addSkillsToVolunteer = (volunteer\_id, skill\_ids, callback) => {
* const values = skill\_ids.map(skill\_id => [volunteer\_id, skill\_id]);
* const sql = 'INSERT IGNORE INTO volunteer\_skills (volunteer\_id, skill\_id) VALUES ?';
* db.query(sql, [values], callback);
* };
* **sponsorModel.js**
* const db = require('../config/db');
* const Sponsor = {
* getAll: () => {
* return new Promise((resolve, reject) => {
* const query = `
* SELECT s.id, s.sponsorship\_type, u.name, u.email, u.phone, u.location
* FROM sponsors s
* JOIN users u ON s.user\_id = u.user\_id
* `;
* db.query(query, (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* },
* getById: (id) => {
* return new Promise((resolve, reject) => {
* const query = `
* SELECT s.\*, u.name AS user\_name, u.email, u.phone, u.location ,u.status
* FROM sponsors s
* JOIN users u ON s.user\_id = u.user\_id
* WHERE s.id = ?
* `;
* db.query(query, [id], (err, results) => {
* if (err) reject(err);
* else resolve(results[0]);
* });
* });
* },
* update: (id, data) => {
* return new Promise((resolve, reject) => {
* db.query('UPDATE sponsors SET ? WHERE id = ?', [data, id], (err, result) => {
* if (err) reject(err);
* else resolve(result);
* });
* });
* },
* };
* module.exports = Sponsor;
* **sponsorshipModel.js**
* const db = require('../config/db');
* const Sponsorship = {
* getAll: () => {
* const query = `
* SELECT s.\*, o.name AS orphan\_name, u.name AS sponsor\_name
* FROM sponsorships s
* LEFT JOIN orphans o ON s.orphan\_id = o.id
* LEFT JOIN sponsors sp ON s.sponsor\_id = sp.id
* LEFT JOIN users u ON sp.user\_id = u.user\_id
* `;
* return new Promise((resolve, reject) => {
* db.query(query, (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* },
* getByOrphanId: (orphanId) => {
* const query = `
* SELECT s.\*, o.name AS orphan\_name, u.name AS sponsor\_name
* FROM sponsorships s
* LEFT JOIN orphans o ON s.orphan\_id = o.id
* LEFT JOIN sponsors sp ON s.sponsor\_id = sp.id
* LEFT JOIN users u ON sp.user\_id = u.user\_id
* WHERE s.orphan\_id = ?
* `;
* return new Promise((resolve, reject) => {
* db.query(query, [orphanId], (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* },
* getBySponsorId: (sponsorId) => {
* const query = `
* SELECT s.\*, o.name AS orphan\_name, u.name AS sponsor\_name
* FROM sponsorships s
* LEFT JOIN orphans o ON s.orphan\_id = o.id
* LEFT JOIN sponsors sp ON s.sponsor\_id = sp.id
* LEFT JOIN users u ON sp.user\_id = u.user\_id
* WHERE s.sponsor\_id = ?
* `;
* return new Promise((resolve, reject) => {
* db.query(query, [sponsorId], (err, results) => {
* if (err) reject(err);
* else resolve(results);
* });
* });
* },
* create: (data) => {
* return new Promise((resolve, reject) => {
* db.query('INSERT INTO sponsorships SET ?', data, (err, result) => {
* if (err) reject(err);
* else resolve(result.insertId);
* });
* });
* },
* update: (id, data) => {
* return new Promise((resolve, reject) => {
* db.query('UPDATE sponsorships SET ? WHERE id = ?', [data, id], (err, result) => {
* if (err) reject(err);
* else resolve(result);
* });
* });
* },
* updateStatus: (id, status) => {
* return new Promise((resolve, reject) => {
* const query = 'UPDATE sponsorships SET status = ? WHERE id = ?';
* db.query(query, [status, id], (err, result) => {
* if (err) reject(err);
* else resolve(result);
* });
* });
* },
* softDelete: (id) => {
* return new Promise((resolve, reject) => {
* const query = 'UPDATE sponsorships SET status = "Not Available" WHERE id = ?';
* db.query(query, [id], (err) => {
* if (err) reject(err);
* else resolve();
* });
* });
* }
* };
* module.exports = Sponsorship;
* **userModel.js**
* const db = require('../config/db');
* const User = {
* create: (data) => {
* return new Promise((resolve, reject) => {
* const query = 'INSERT INTO users SET ?';
* db.query(query, data, (err, result) => {
* if (err) reject(err);
* else resolve(result.insertId);
* });
* });
* },
* insertToRoleTable: (role, userId) => {
* return new Promise((resolve, reject) => {
* let roleTable = '';
* switch (role) {
* case 'sponsor':
* roleTable = 'sponsors';
* break;
* case 'donor':
* roleTable = 'donors';
* break;
* case 'volunteer':
* roleTable = 'volunteers';
* break;
* case 'delivery':
* roleTable = 'delivery';
* break;
* case 'orphanages':
* roleTable = 'orphanages';
* break;
* default:
* return reject(new Error('Invalid role'));
* }
* const query = `INSERT INTO ${roleTable} (user\_id) VALUES (?)`;
* db.query(query, [userId], (err, result) => {
* if (err) reject(err);
* else resolve(result.insertId);
* });
* });
* },
* getAllUsers: () => {
* return new Promise((resolve, reject) => {
* const query = `
* SELECT user\_id, name, email, phone, location, role, status, created\_at
* FROM users
* `;
* db.query(query, (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* },
* getUserById: (id) => {
* return new Promise((resolve, reject) => {
* const query = `
* SELECT user\_id, name, email, phone, location, role, status, created\_at
* FROM users
* WHERE user\_id = ?
* `;
* db.query(query, [id], (err, results) => {
* if (err) return reject(err);
* resolve(results[0]);
* });
* });
* },
* getUsersByRole: (role) => {
* return new Promise((resolve, reject) => {
* const query = `
* SELECT user\_id, name, email, phone, location, role, status, created\_at
* FROM users
* WHERE role = ?
* `;
* db.query(query, [role], (err, results) => {
* if (err) return reject(err);
* resolve(results);
* });
* });
* },
* update: (id, data) => {
* return new Promise((resolve, reject) => {
* const query = 'UPDATE users SET ? WHERE user\_id = ?';
* db.query(query, [data, id], (err, result) => {
* if (err) return reject(err);
* resolve(result);
* });
* });
* },
* softDelete: (id) => {
* return new Promise((resolve, reject) => {
* const query = 'UPDATE users SET status = ? WHERE user\_id = ?';
* db.query(query, ['Suspended', id], (err, result) => {
* if (err) return reject(err);
* resolve(result);
* });
* });
* },
* };
* module.exports = User;
* **volunteerModel.js**
* const db = require('../config/db');
* exports.getAllVolunteers = () => {
* return new Promise((resolve, reject) => {
* const sql = `
* SELECT v.volunteer\_id, v.availability, v.user\_id,
* s.skill\_id, s.name AS skill\_name
* FROM volunteers v
* LEFT JOIN volunteer\_skills vs ON v.volunteer\_id = vs.volunteer\_id
* LEFT JOIN skills s ON vs.skill\_id = s.skill\_id
* ORDER BY v.volunteer\_id ASC
* `;
* db.query(sql, (err, results) => {
* if (err) return reject(err);
* const volunteersMap = new Map();
* for (const row of results) {
* const vid = row.volunteer\_id;
* if (!volunteersMap.has(vid)) {
* volunteersMap.set(vid, {
* volunteer\_id: row.volunteer\_id,
* availability: row.availability,
* user\_id: row.user\_id,
* skills: []
* });
* }
* if (row.skill\_id) {
* volunteersMap.get(vid).skills.push({
* skill\_id: row.skill\_id,
* name: row.skill\_name
* });
* }
* }
* resolve([...volunteersMap.values()]);
* });
* });
* };
* **routes**
  + authRoutes.js
* const express = require('express');
* const router = express.Router();
* const authController = require('../controllers/authController');
* router.post('/login', authController.login);
* module.exports = router;
  + deliveryTrackingRoutes.js
* // routes/deliveryTrackingRoutes.js
* const express = require('express');
* const router = express.Router();
* const deliveryTrackingController = require('../controllers/deliveryTrackingController');
* // POST /api/delivery-tracking/assign
* router.post('/assign', deliveryTrackingController.assignDelivery);
* router.post('/confirm', deliveryTrackingController.confirmDelivery);
* module.exports = router;
  + donationPaymentsRoutes.js
* const express = require('express');
* const router = express.Router();
* const controller = require('../controllers/donationPaymentController');
* router.get('/', controller.getAllPayments);
* router.get('/summary', controller.getSummary);
* router.get('/:id', controller.getPaymentById);
* router.post('/', controller.createPayment);
* router.delete('/:id', controller.deletePayment);
* router.put('/update-fee-percent', controller.updateTransactionFeePercent);
* module.exports = router;
  + donationReqRoutes.js
* const express = require('express');
* const router = express.Router();
* const logisticsController = require('../controllers/donationreqController');
* // GET /api/notifications
* router.get('/', logisticsController.getAllLogisticReq);
* router.post('/', logisticsController.createLogisticReq);
* module.exports = router;
  + donationRoutes.js
* // routes/donation.routes.js
* const express = require('express');
* const router = express.Router();
* const donationController = require('../controllers/donationController.js');
* // POST /api/donations
* router.post('/', donationController.createDonation);
* // GET /api/donations
* router.get('/', donationController.getAllDonations);
* router.put('/:donation\_id', donationController.updateDonationFields);
* router.get('/NoImpact', donationController.getDonationsNoImpact);
* module.exports = router;
  + humanRequest.routes.js
* const express = require('express');
* const router = express.Router();
* const humanRequestController = require('../controllers/humanRequest.controller');
* router.post('/orphanages/:orphanage\_id', humanRequestController.createRequest);
* router.get('/', humanRequestController.getAllRequests);
* module.exports = router;
  + logisticsRoutes.js
* const express = require('express');
* const router = express.Router();
* const controller = require('../controllers/logisticsController');
* router.get('/', controller.getAllTasks);
* router.get('/:id', controller.getTaskById);
* router.post('/', controller.createTask);
* router.put('/:id', controller.updateTask);
* router.delete('/:id', controller.deleteTask);
* module.exports = router;
  + match.routes.js
* const express = require('express');
* const router = express.Router();
* const controller = require('../controllers/match.controller');
* // Get volunteers who match a specific request
* router.get('/:request\_id', controller.getMatchingVolunteers);
* router.put('/orphanagesupdate/:request\_id', controller.updateRequestStatusAndNotify);
* router.put('/volunteersupdate/:request\_id', controller.volunteerRespondToRequest);
* module.exports = router;
  + notificationRoutes.js
* const express = require('express');
* const router = express.Router();
* const notificationController = require('../controllers/notificationController');
* // GET /api/notifications
* router.get('/', notificationController.getAllNotifications);
* router.get('/:donor\_id', notificationController.getNotificationsByDonor);
* router.post('/send-emergency', notificationController.sendEmergencyNotificationsToAllUsers);
* module.exports = router;
  + orphanRoutes.js
* const express = require('express');
* const router = express.Router();
* const orphanController = require('../controllers/orphanController');
* router.get('/', orphanController.getAllOrphans);
* router.get('/:id', orphanController.getOrphanById);
* router.post('/', orphanController.createOrphan);
* router.put('/:id', orphanController.updateOrphan);
* router.delete('/:id', orphanController.softDeleteOrphan);
* module.exports = router;
  + orphanUpdateRoutes.js
* const express = require('express');
* const router = express.Router();
* const controller = require('../controllers/orphanUpdateController');
* const multer = require('multer');
* const path = require('path');
* // إعداد التخزين للصور
* const storage = multer.diskStorage({
* destination: (req, file, cb) => {
* cb(null, 'images/');
* },
* filename: (req, file, cb) => {
* const uniqueName = Date.now() + '-' + file.originalname;
* cb(null, uniqueName);
* },
* });
* const upload = multer({ storage });
* // المسارات
* router.get('/', controller.getAllUpdates);
* router.get('/:id', controller.getUpdatesByOrphan);
* router.post('/', upload.single('image'), controller.createUpdate);
* router.put('/:id', upload.single('image'), controller.updateUpdate);
* router.delete('/:id', controller.deleteUpdate);
* module.exports = router;
  + partnerRoutes.js
* const express = require('express');
* const router = express.Router();
* const controller = require('../controllers/partnerController');
* router.get('/', controller.getAllPartners);
* router.get('/:id', controller.getPartnerById);
* router.post('/', controller.createPartner);
* router.put('/:id', controller.updatePartner);
* router.delete('/:id', controller.deletePartner);
* module.exports = router;
  + skill.routes.js
* const express = require('express');
* const router = express.Router();
* const skillController = require('../controllers/skill.controller');
* router.post('/', skillController.createSkill);
* router.get('/', skillController.getAllSkills);
* router.post('/:volunteer\_id', skillController.addSkills);
* module.exports = router;
  + sponsorRoutes.js
* const express = require('express');
* const router = express.Router();
* const sponsorController = require('../controllers/sponsorController');
* const verifyToken = require('../middlewares/verifyToken');
* const checkPermission = require('../middlewares/checkPermission');
* router.post(
* '/external/export-sheet',
* verifyToken,
* checkPermission('canViewSponsors'),
* sponsorController.exportSponsorsToGoogleSheet
* );
* router.post(
* '/external/post-by-id/:id',
* verifyToken,
* checkPermission('canViewSponsors'),
* sponsorController.postSponsorByIdToExternalAPI
* );
* router.get('/', sponsorController.getAllSponsors);
* router.get('/:id', sponsorController.getSponsorById);
* router.put('/:id', sponsorController.updateSponsor);
* module.exports = router;
  + sponsorshipRoutes.js
* const express = require('express');
* const router = express.Router();
* const sponsorshipController = require('../controllers/sponsorshipController');
* router.get('/', sponsorshipController.getAllSponsorships);
* router.get('/orphan/:orphanId', sponsorshipController.getByOrphanId);
* router.get('/sponsor/:sponsorId', sponsorshipController.getBySponsorId);
* router.post('/', sponsorshipController.createSponsorship);
* router.put('/:id', sponsorshipController.updateSponsorship);
* router.put('/:id/status', sponsorshipController.updateSponsorshipStatus);
* router.delete('/:id', sponsorshipController.softDeleteSponsorship);
* module.exports = router;
  + userRoutes.js
* const express = require('express');
* const router = express.Router();
* const userController = require('../controllers/userController');
* router.post('/', userController.createUser);
* router.get('/', userController.getAllUsers);
* router.get('/:id', userController.getUserById);
* router.get('/role/:role', userController.getUsersByRole);
* router.put('/:id', userController.updateUser);
* router.delete('/:id', userController.softDeleteUser);
* module.exports = router;
  + volunteerRoutes.js
* const express = require('express');
* const router = express.Router();
* const volunteerController = require('../controllers/volunteerController.js');
* // GET /api/donations
* router.get('/', volunteerController.getAllVolunteers);
* module.exports = router;
* **Env**
* # ✅ إعدادات JWT
* JWT\_SECRET=your\_secret\_key
* # ✅ إعدادات السيرفر
* PORT=3000
* # ✅ إعدادات قاعدة البيانات (MySQL)
* DB\_HOST=localhost
* DB\_USER=root
* DB\_PASSWORD=
* DB\_NAME=hopeconnect\_db
* # ✅ إعدادات Google Sheets (مسار ملف الحساب الخدمي)
* GOOGLE\_APPLICATION\_CREDENTIALS=./config/service-account.json
* **server.js**
* const express = require('express');
* const bodyParser = require('body-parser');
* const db = require('./config/db');
* require('dotenv').config();
* const app = express();
* const PORT = process.env.PORT || 3000;
* app.use(bodyParser.json());
* app.use('/api/images', express.static('images'));
* const orphanRoutes = require('./routes/orphanRoutes');
* const sponsorRoutes = require('./routes/sponsorRoutes');
* const sponsorshipRoutes = require('./routes/sponsorshipRoutes');
* const orphanUpdateRoutes = require('./routes/orphanUpdateRoutes');
* const userRoutes = require('./routes/userRoutes');
* const authRoutes = require('./routes/authRoutes');
* const donationPaymentsRoutes = require('./routes/donationPaymentsRoutes');
* const partnersRoutes = require('./routes/partnerRoutes');
* const deliveryTrackingRoutes = require('./routes/deliveryTrackingRoutes');
* const logisticsTasksRoutes = require('./routes/logisticsRoutes');
* const donationRoutes = require('./routes/donationRoutes.js');
* const notificationRoutes = require('./routes/notificationRoutes.js');
* const donationReqs = require('./routes/donationReqRoutes.js');
* const volunteer = require('./routes/volunteerRoutes.js');
* const skillRoutes = require('./routes/skill.routes');
* const humanRequestRoutes = require('./routes/humanRequest.routes');
* const matchRoutes = require('./routes/match.routes');
* app.use('/api/orphans', orphanRoutes);
* app.use('/api/sponsors', sponsorRoutes);
* app.use('/api/sponsorships', sponsorshipRoutes);
* app.use('/api/orphan-updates', orphanUpdateRoutes);
* app.use('/api/users', userRoutes);
* app.use('/api', authRoutes);
* app.use('/api/donation-payments', donationPaymentsRoutes);
* app.use('/api/partners', partnersRoutes);
* app.use('/api/delivery-tracking', deliveryTrackingRoutes);
* app.use('/api/logistics-tasks', logisticsTasksRoutes);
* app.use('/images', express.static('images'));
* app.use('/api/donations', donationRoutes);
* app.use('/api/notifications', notificationRoutes);
* app.use('/api/donationReqs', donationReqs);
* app.use('/api/volunteers', volunteer);
* app.use('/api/skills', skillRoutes);
* app.use('/api/volunteer-skills', skillRoutes);
* app.use('/api/requests', humanRequestRoutes);
* app.use('/api/match', matchRoutes);
* app.get('/', (req, res) => {
* res.send('HopeConnect service is running');
* });
* app.listen(PORT, () => {
* console.log(`Server is running on http://localhost:${PORT}`);
* });