**File: allowedTo.js**

const appError = require("../utilities/appError");

const httpStatusText = require('../utilities/httpStatusText.js');

module.exports = (...roles) => {

return (req, res, next) => {

if (!roles.includes(req.currentUser.role)) {

return next(appError.create('Your forbidden to do this action'), 403, httpStatusText.ERROR);

}

next();

}

}

**File: asyncWrapper.js**

// to handle try-catch stmts

module.exports = (asyncFunc) => {

return (req, res, next) => { // returns middleware

asyncFunc(req, res, next).catch((err) => { // catchs any error happen in asyncFunc

next(err); /\*runs next middleware\*/ // goes to meddleware3 in index.js

}); // .catch() can only be used with async functions

}

};

**File: checkOrphanage.js**

const mongoose = require("mongoose");

const Orphan = require("../models/orphan.model.js");

const Orphanage = require("../models/orphanage.model.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const asyncWrapper = require('../middlewares/asyncWrapper.js');

const requestStatus = require("../utilities/requestStatus");

const userRoles = require("../utilities/userRoles.js");

const checkOrphanage = asyncWrapper(

async (req, res, next) => {

if(req.currentUser.role === userRoles.ORPHANAGE\_ADMIN){

let orphanageId;

let orphanId;

let oldOrphan;

if(Object.keys(req.params).length === 0){ // on add new orphan

orphanageId = req.body.orphanage;

}

else{ // on get operations

if(req.params.orphanid){

orphanId = req.params.orphanid;

oldOrphan = await Orphan.findById(orphanId);

if (!oldOrphan) {

return next(appError.create("Orphan not found", 404, httpStatusText.FAIL));

}

}

else if(req.params.orphanageid){

orphanageId = req.params.orphanageid;

}

else if(req.params.id){

orphanageId = req.params.id;

}

}

const orphanageAdmin = req.currentUser.id;

let orphanageDoc;

if(req.params.orphanid){

orphanageDoc = await Orphanage.findById(new mongoose.Types.ObjectId(oldOrphan.orphanage));

}

else{

orphanageDoc = await Orphanage.findById(new mongoose.Types.ObjectId(orphanageId));

}

if (!orphanageDoc) {

const error = appError.create("Orphanage not found", 400, httpStatusText.FAIL);

return next(error);

}

if (orphanageDoc.status !== requestStatus.APPROVED){

const error = appError.create("Orphanage not approved", 400, httpStatusText.FAIL);

return next(error);

}

if (!orphanageDoc.admin.equals(orphanageAdmin)) {

const error = appError.create("Orphanage Admin mismatch", 400, httpStatusText.FAIL);

return next(error);

}

}

next();

return;

}

);

module.exports = checkOrphanage;

**File: controlDonation.controller.js**

const ControllingDonation = require("../models/controllingDonation.model.js");

const Donation = require("../models/donation.model.js");

const User = require("../models/user.model.js");

const asyncWrapper = require("../middlewares/asyncWrapper.js");

const Orphan = require("../models/orphan.model.js");

const userRoles = require('../utilities/userRoles');

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

// إنشاء سجل تحكم جديد (Control Donation)

const mongoose = require("mongoose");

const Orphanage = require("../models/orphanage.model.js"); // تأكد الاستيراد في أعلى الملف

const createControlDonation = asyncWrapper(async (req, res, next) => {

const { donationId } = req.params;

// 1. البحث عن التبرع

const donation = await Donation.findById(donationId);

if (!donation) {

return next(appError.create("Donation not found", 404, httpStatusText.FAIL));

}

if (donation.status !== "Completed") {

return next(appError.create("Donation must be completed before controlling", 400, httpStatusText.FAIL));

}

const userId = req.currentUser.id;

const userRole = req.currentUser.role;

// --- طباعة بيانات للتتبع ---

console.log("User ID:", userId);

console.log("User role:", userRole);

console.log("Donation orphanage ID:", donation.orphanage);

// 2. تحقق الصلاحيات اعتمادًا على دور المستخدم وربطه بمؤسسة التبرع

if (donation.orphanage) {

const orphanage = await Orphanage.findById(donation.orphanage);

console.log("Orphanage admin ID:", orphanage ? orphanage.admin : "No orphanage found");

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

if (userRole === "ORPHANAGE ADMIN") {

if (!orphanage.admin.equals(userId)) {

return next(appError.create("Unauthorized - you are not admin of this orphanage", 403, httpStatusText.FAIL));

}

} else if (userRole !== "ADMIN") {

return next(appError.create("Unauthorized", 403, httpStatusText.FAIL));

}

} else {

// حالة التبرع بدون orphanage مرتبط - فقط Admin يسمح له

if (userRole !== "ADMIN") {

return next(appError.create("Unauthorized", 403, httpStatusText.FAIL));

}

}

const { usageSummary, orphansImpacted, photos, notes } = req.body;

// تحويل أسماء ملفات الصور إلى روابط كاملة

let photosUrls = [];

if (Array.isArray(photos) && photos.length > 0) {

photosUrls = photos.map(filename => `http://localhost:5000/uploads/${filename}`);

}

// 3. تحقق إن سجل التحكم ما موجود مسبقاً (لمنع التعديل)

const existingRecord = await ControllingDonation.findOne({ donation: donationId });

if (existingRecord) {

return next(appError.create("Control record already exists", 400, httpStatusText.FAIL));

}

// 4. إنشاء سجل التحكم الجديد

const controllingRecord = new ControllingDonation({

donation: donationId,

orphanage: donation.orphanage || null,

controlledBy: userId,

usageSummary,

orphansImpacted: orphansImpacted || [],

photos: photosUrls,

notes: notes || ""

});

await controllingRecord.save();

donation.status = "Controlled";

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Donation control record created successfully",

data: { controllingRecord }

});

});

module.exports = {

createControlDonation,

};

const getControlDonationById = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const record = await ControllingDonation.findById(id)

.populate("donation")

.populate("orphanage")

.populate("controlledBy", "name email")

.populate("orphansImpacted", "name age");

if (!record) {

return next(appError.create("Control donation record not found", 404, httpStatusText.FAIL));

}

const userId = req.currentUser.id;

const userRole = req.currentUser.role;

if (record.orphanage) {

const orphanage = await Orphanage.findById(record.orphanage);

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

if (userRole === "ORPHANAGE ADMIN") {

if (!orphanage.admin.equals(userId)) {

return next(appError.create("Unauthorized - you are not admin of this orphanage", 403, httpStatusText.FAIL));

}

} else if (userRole !== "ADMIN") {

return next(appError.create("Unauthorized", 403, httpStatusText.FAIL));

}

} else {

// حالة سجل التحكم بدون orphanage مرتبط - فقط Admin يسمح له

if (userRole !== "ADMIN") {

return next(appError.create("Unauthorized", 403, httpStatusText.FAIL));

}

}

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { record }

});

});

const getControlDonations = asyncWrapper(async (req, res, next) => {

const userId = req.currentUser.id;

const userRole = req.currentUser.role;

let records;

if (userRole === "ADMIN") {

// المدير يشوف كل السجلات

records = await ControllingDonation.find()

.populate("donation")

.populate("orphanage")

.populate("controlledBy", "name email")

.populate("orphansImpacted", "name age");

} else if (userRole === "ORPHANAGE ADMIN") {

// مشرف المؤسسة يشوف السجلات الخاصة بمؤسسته فقط

const orphanages = await Orphanage.find({ admin: userId });

const orphanageIds = orphanages.map(o => o.\_id);

records = await ControllingDonation.find({ orphanage: { $in: orphanageIds } })

.populate("donation")

.populate("orphanage")

.populate("controlledBy", "name email")

.populate("orphansImpacted", "name age");

} else {

return next(appError.create("Unauthorized", 403, httpStatusText.FAIL));

}

res.status(200).json({

status: httpStatusText.SUCCESS,

results: records.length,

data: { records }

});

});

const updateControlDonation = asyncWrapper(async (req, res, next) => {

const { id } = req.params; // معرف سجل التحكم المراد تحديثه

const userId = req.currentUser.id;

const userRole = req.currentUser.role;

const updateData = req.body; // البيانات اللي حابب تحدثها

// 1. جلب سجل التحكم

const record = await ControllingDonation.findById(id);

if (!record) {

return next(appError.create("Control donation record not found", 404, httpStatusText.FAIL));

}

// 2. تحقق الصلاحيات بناءً على الدور والارتباط بالـ orphanage

if (record.orphanage) {

const orphanage = await Orphanage.findById(record.orphanage);

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

if (userRole === "ORPHANAGE ADMIN" && !orphanage.admin.equals(userId)) {

return next(appError.create("Unauthorized - you are not admin of this orphanage", 403, httpStatusText.FAIL));

}

// 3. تحديث الحقول المسموح بها فقط (لتجنب التحديثات غير المرغوبة)

const allowedUpdates = ["usageSummary", "orphansImpacted", "photos", "notes"];

allowedUpdates.forEach(field => {

if (updateData[field] !== undefined) {

record[field] = updateData[field];

}

});

// 4. حفظ التحديثات

await record.save();

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Control donation record updated successfully",

data: { record }

});

}});

module.exports = {

createControlDonation,

getControlDonationById,

getControlDonations,

updateControlDonation,

};

**File: controlDonation.route.js**

const express = require("express");

const router = express.Router();

const controllingDonationController = require("../controllers/controlDonation.controller.js");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

// إنشاء سجل تحكم

router.post(

"/:donationId/control",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN, userRoles.ADMIN),

controllingDonationController.createControlDonation

);

// جلب سجل معين بواسطة ID

router.get(

"/:id",

verifyToken,

allowedTo(userRoles.ADMIN, userRoles.ORPHANAGE\_ADMIN),

controllingDonationController.getControlDonationById

);

router.get("/", verifyToken, allowedTo(userRoles.ADMIN, userRoles.ORPHANAGE\_ADMIN), controllingDonationController.getControlDonations);

router.patch(

"/:id",

verifyToken,

allowedTo(userRoles.ADMIN, userRoles.ORPHANAGE\_ADMIN),

controllingDonationController.updateControlDonation

);

module.exports = router;

**File: controllingDonation.model.js**

const mongoose = require("mongoose");

const ControllingDonationSchema = new mongoose.Schema({

donation: { type: mongoose.Schema.Types.ObjectId, ref: "Donation", required: true, unique: true },

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage" },

controlledBy: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

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

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

orphansImpacted: [{ type: mongoose.Schema.Types.ObjectId, ref: "Orphan" }],

photos: [{ type: String }],

notes: { type: String },

});

module.exports = mongoose.model("ControllingDonation", ControllingDonationSchema);

**File: course.model.js**

const mongoose = require('mongoose');

const courseSchema = new mongoose.Schema({

name: {

type: String,

required: true

},

price: {

type: Number,

required: true

}

});

module.exports = mongoose.model('Course', courseSchema);

**File: courses.controller.js**

// ../ => to move one step back in the path

const { json } = require('express');

let Course = require('../models/course.model.js');

const httpStatusText = require('../utilities/httpStatusText.js');

const {validationResult} = require('express-validator');

const asyncWrapper = require('../middlewares/asyncWrapper.js');

const appError = require('../utilities/appError.js');

// Course.find(querryObject, projection);

// querryObject:

// Course.find({name: 'Math', price: 100}); //find me the course with name "math" and price 100

// Course.find({price: 100}); //find me the course with price 100

// Course.find({price: {$gt: 500}}); //find me the course with price > 500

// projection:

// Course.find({name: 'Math', price: 100}, {name: 1}); // 1 or true

// means: find me the course with name "math" and price 100, and return only the name

// Course.find({name: 'Math', price: 100}, {\_\_id: false}); // 0 or false

// means: find me the course with name "math" and price 100, and return all fields except the id field

const getAllCourses = asyncWrapper(

async (req,res) => {

const query = req.query; // to get the data in the url after '?'

/\* Pagination \*/

// max number of elements in one page

const limit = query.limit || 4; // means that default limit is 4 (if it not specified in the url)

// current page

const page = query.page || 1; // means that default page is 1 (if it not specified in the url)

const skip = (page - 1) \* limit; // skip the first ((page - 1) \* limit) elements

// get all courses from db using Course model

const courses = await Course.find({} /\*return all\*/, {\_\_v: false}).limit(limit).skip(skip);

/\* Pagination \*/

// res.send();

return res.json({ status: httpStatusText.SUCCESS, data: {courses} });

}

);

const getCourseByID = asyncWrapper(

async (req, res, next) => { // the parameter feature(:id) is provided by express

// const id = req.params.id; // req.params.id => taken from url (:id) => string

// const course = courses.find((course) => course.id === parseInt(id));

const course = await Course.findById(req.params.id);

if(!course){

// const error = new Error();

// error.message = "course not found";

// error.statusCode = 404;

const error = appError.create("course not found", 400, httpStatusText.FAIL);

return next(error);

// return res.status(404).json({ status: httpStatusText.FAIL, data: {course: "course not found" /\*or null\*/} }); // return to stop executing func

}

return res.json({ status: httpStatusText.SUCCESS,data: {course} });

// try{

// }

// catch(err){

// return res.status(400).json({ status: httpStatusText.ERROR, data: null, message: "Invalid Object ID" /\*or err.message\*/, code: 400});

// }

}

);

const addNewCourse = asyncWrapper(

async (req, res, next) => {

// // data validation

// if(!req.body.name){

// return res.status(400).json({ message: 'Name is required' });

// }

// else if(!req.body.price){

// return res.status(400).json({ message: 'Price is required' });

// }

// // type validation

// else if(typeof req.body.name !== 'string'){

// return res.status(400).json({ message: 'Name must be a string' });

// }

// else if(typeof req.body.price !== 'number'){

// return res.status(400).json({ message: 'Price must be a number' });

// }

// or using express validator (in post parameters above) or joi js or zod dev

const errors = validationResult(req);

if (!errors.isEmpty()) {

// const error = new Error();

// error.message = "Invalid request data";

// error.statusCode = 400;

const error = appError.create(errors.array(), 400, httpStatusText.FAIL);

return next(error);

// return res.status(400).json({ status: httpStatusText.FAIL, data: {errors: errors.array()} });

}

const newCourse = new Course(req.body);

await newCourse.save();

// created

res.status(201).json({ status: httpStatusText.SUCCESS, data: {course: newCourse} });

// const newCourse = {

// id: courses.length + 1,

// name: req.body.name, // cant be accessed(undefined) without bodyparser middleware

// price: req.body.price // cant be accessed(undefined) without bodyparser middleware

// // or

// // ...req.body // (put all body info)

// };

// courses.push(newCourse);

}

);

const updateCourse = asyncWrapper(

async (req, res) => {

// id is a parameter

const id = req.params.id;

// let course = courses.find((course) => course.id === parseInt(id));

// const course = await Course.findByIdAndUpdate(id, {$set: {...req.body}});// returns the found document - not the updated - then update it

const updatedCourse = await Course.updateOne({\_id: id}, {$set: {...req.body}});

// if(!course){

// return res.status(404).json({ message: 'Course not found' }); // return to stop executing func

// }

// course = {

// ...course, // spread operator (put old course values)

// ...req.body // (put new values - overrides (merging) the old ones)

// }

// courses[id - 1] = course;

return res.status(200).json({ status: httpStatusText.SUCCESS, data: {course: updatedCourse} });

// try{

// }

// catch(err){

// return res.status(400).json({ status: httpStatusText.ERROR, data: null, message: err.message, code: 400 });

// }

}

);

const deleteCourse = asyncWrapper(

async (req, res) => {

const id = req.params.id;

// let course = courses.find((course) => course.id === parseInt(id));

await Course.deleteOne({\_id: id});

// if(!course){

// return res.status(404).json({ message: 'Course not found' });

// }

// courses.splice(id - 1, 1); // or

// //courses = courses.filter((course) => course.id !== parseInt(id)); // return all courses that thier ids != deleted course id

// // fixing ids of courses array:

// courses.forEach((course, index) => {

// course.id = index + 1;

// });

return res.status(200).json({ success: httpStatusText.SUCCESS, data: null });

// try{

// }

// catch(err){

// return res.status(400).json({ status: httpStatusText.ERROR, data: null, message: err.message, code: 400 });

// }

}

);

module.exports = {

getAllCourses,

getCourseByID,

addNewCourse,

updateCourse,

deleteCourse

};

**File: courses.route.js**

const express = require('express');

const router = express.Router(); // mini app (instead of app)

const coursesController = require('../controllers/courses.controller.js');

const {validationSchema} = require('../middlewares/validationSchema.js');

const verifyToken = require('../middlewares/verifyToken.js');

const userRoles = require('../utilities/userRoles.js');

const allowedTo = require('../middlewares/allowedTo.js');

// CRUD (Create / Read / Update / Delete)

// dont make the naming of you apis complex like:

// /api/get-all-courses

// /api/get-course-by-id

// /api/add-new-course

// because:

// 1. it's not a good practice

// 2. it's not RESTful

// 3. it's not scalable

// 4. it's not maintainable

// 5. it's not easy to understand

// 6. it's not easy to use

// 7. it's not easy to test

// 8. it's not easy to document

// 9. it's not easy to debug

// 10. it's not easy to refactor

// instead, use simple and clear naming like:

// /courses

//Routes (Resources) and there handlers (callbacks - controllers)

router.route('/')

// get all courses

.get(coursesController.getAllCourses) // get: takes data from server to client

// add new course: using request body

.post(/\*middlewares-handlers\*/verifyToken, allowedTo(userRoles.ADMIN), validationSchema(), coursesController.addNewCourse); // post: takes data from client to server

router.route('/:id')

// get single course

.get(coursesController.getCourseByID)

// update a course:

.patch(validationSchema(), coursesController.updateCourse)// put: replaces the object you want to update with the new object // patch: update only one value (for example title of js or price of c++)

// delete a course:

.delete(verifyToken, allowedTo(userRoles.ADMIN) , coursesController.deleteCourse); // delete: removes the object you want to delete

module.exports = router;

**File: dashboard.controller.js**

const Donation = require("../models/donation.model.js");

const ControllingDonation = require("../models/controllingDonation.model.js");

const asyncWrapper = require("../middlewares/asyncWrapper.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

// 1. جلب كل سجلات التحكم المرتبطة بتبرعات الدونور

const getDonorControlRecords = asyncWrapper(async (req, res, next) => {

const donorId = req.currentUser.id;

const donations = await Donation.find({ donor: donorId }).select("\_id");

if (!donations || donations.length === 0) {

return next(appError.create("No donations found for this donor", 404, httpStatusText.FAIL));

}

const donationIds = donations.map(d => d.\_id);

const controlRecords = await ControllingDonation.find({ donation: { $in: donationIds } })

// .populate("donation")

.populate("orphanage", "name location")

.populate("controlledBy", "name email")

.populate("orphansImpacted", "name age");

res.status(200).json({

status: httpStatusText.SUCCESS,

results: controlRecords.length,

data: { controlRecords }

});

});

// 2. جلب سجل تحكم معين للدونور بالمعرف

const getDonorControlRecordsById = asyncWrapper(async (req, res, next) => {

const donorId = req.currentUser.id;

const { id } = req.params;

const record = await ControllingDonation.findById(id)

// .populate("donation")

.populate("orphanage", "name location")

.populate("controlledBy", "name email")

.populate("orphansImpacted", "name age");

if (!record) {

return next(appError.create("Control donation record not found", 404, httpStatusText.FAIL));

}

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { record }

});

});

const getDonorSummary = asyncWrapper(async (req, res, next) => {

const donorId = req.currentUser.id;

// جلب كل التبرعات للدونور

const donations = await Donation.find({ donor: donorId });

if (!donations || donations.length === 0) {

return next(appError.create("No donations found for this donor", 404, httpStatusText.FAIL));

}

// استخراج الـ IDs وأنواع التبرعات

const donationIds = donations.map(d => d.\_id);

const donationTypes = {};

donations.forEach(donation => {

donationTypes[donation.\_id.toString()] = donation.donationType;

});

// جلب سجلات التحكم المرتبطة بهذه التبرعات

const controlRecords = await ControllingDonation.find({ donation: { $in: donationIds } }).select("donation");

// عد سجلات التحكم لكل نوع تبرع

const controlCountByType = {};

controlRecords.forEach(record => {

const donationType = donationTypes[record.donation.toString()];

if (donationType) {

controlCountByType[donationType] = (controlCountByType[donationType] || 0) + 1;

}

});

// عد التبرعات لكل نوع

const donationCountByType = donations.reduce((acc, donation) => {

acc[donation.donationType] = (acc[donation.donationType] || 0) + 1;

return acc;

}, {});

res.status(200).json({

status: httpStatusText.SUCCESS,

data: {

donationCountByType,

controlCountByType

}

});

});

module.exports = {

getDonorControlRecords,

getDonorControlRecordsById,

getDonorSummary,

};

**File: dashboard.route.js**

const express = require("express");

const router = express.Router();

const dashboardController = require("../controllers/dashboard.controller.js");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

// مسار لجلب كل سجلات التحكم المرتبطة بتبرعات الدونور

router.get(

"/donor/control-records",

verifyToken,

allowedTo(userRoles.DONOR),

dashboardController.getDonorControlRecords

);

// مسار لجلب سجل تحكم معين للدونور بواسطة ID

router.get(

"/donor/control-records/:id",

verifyToken,

allowedTo(userRoles.DONOR),

dashboardController.getDonorControlRecordsById

);

// مسار لعرض ملخص تبرعات الدونور حسب النوع

router.get(

"/donor/summary",

verifyToken,

allowedTo(userRoles.DONOR),

dashboardController.getDonorSummary

);

module.exports = router;

**File: database.js**

const { Sequelize } = require("sequelize");

const sequelize = new Sequelize(

process.env.DB\_NAME,

process.env.DB\_USER,

process.env.DB\_PASSWORD, {

host: process.env.DB\_HOST,

dialect: "mysql",

logging: false, // Set to true for debugging queries

});

module.exports = sequelize;

**File: deliveryRequest.controller.js**

const DeliveryRequest = require("../models/deliveryRequest.model");

const Donation = require("../models/donation.model");

const User = require("../models/user.model");

const Orphanage = require("../models/orphanage.model");

const asyncWrapper = require("../middlewares/asyncWrapper");

const appError = require("../utilities/appError");

const httpStatusText = require("../utilities/httpStatusText");

const sendEmail = require("../utilities/sendEmail");

const mongoose = require("mongoose");

const userRoles = require("../utilities/userRoles");

// create new delivery request

const createDeliveryRequest = asyncWrapper(async (req, res, next) => {

const donor = req.currentUser;

const { donation, orphanage, location, destination } = req.body;

if (!donation || !location || !destination) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const donationExists = await Donation.findOne({ \_id: donation });

if (!donationExists) return next(appError.create("Donation not found", 404, httpStatusText.FAIL));

if (donationExists.status !== "Pending") return next(appError.create("Donation must be Pending", 400, httpStatusText.FAIL));

if (donationExists.donationType === "Financial") return next(appError.create("Only physical donations allowed", 400, httpStatusText.FAIL));

if (orphanage) {

const orphanageExists = await Orphanage.findById(orphanage);

if (!orphanageExists) return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

if (

!location.coordinates ||

typeof location.coordinates[0] !== "number" || // latitude

typeof location.coordinates[1] !== "number" // longitude

) {

return next(appError.create("your location must have valid coordinates", 400, httpStatusText.FAIL));

}

if (

!destination.coordinates ||

typeof destination.coordinates[0] !== "number" || // latitude

typeof destination.coordinates[1] !== "number" // longitude

) {

return next(appError.create("destination location must have valid coordinates", 400, httpStatusText.FAIL));

}

// Step 1: Send Email First (don't save delivery yet)

const availableDrivers = await User.find({ role: "DRIVER", driverStatus: "AVAILABLE" });

if (!availableDrivers.length) {

return next(appError.create("No available drivers at the moment", 400, httpStatusText.FAIL));

}

// Temporarily create a fake deliveryRequest to get the ID (not saved)

const tempId = new mongoose.Types.ObjectId();

const claimLink = `${process.env.FRONTEND\_URL}/claim-delivery.html?deliveryId=${tempId}`;

const html = `

<h3>🚚 New Delivery Request</h3>

<p>A donor has submitted a delivery request for physical donation pickup.</p>

<p><strong>Pickup Location:</strong> [${location.coordinates.join(", ")}]</p>

<p><a href="${claimLink}">Click here to claim the delivery</a></p>

`;

for (const driver of availableDrivers) {

await sendEmail({

to: driver.email,

subject: "New Delivery Request Available",

html

});

}

// Step 2: Only now save the delivery request

const deliveryRequest = await DeliveryRequest.create({

\_id: tempId,

donor: donor.id,

donation,

orphanage,

status: "PENDING",

location,

destination

});

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Delivery request created and drivers notified.",

data: { deliveryId: deliveryRequest.\_id }

});

});

// Claim delivery request

const claimDeliveryRequest = asyncWrapper(async (req, res, next) => {

const driverId = req.currentUser.id;

const { id } = req.params;

const delivery = await DeliveryRequest.findById(id);

if (!delivery) {

return next(appError.create("Delivery request not found", 404, httpStatusText.FAIL));

}

if (delivery.status !== "PENDING") {

return next(appError.create("Delivery request is no longer available", 400, httpStatusText.FAIL));

}

// Update delivery status to CLAIMED and assign driver

delivery.status = "CLAIMED";

delivery.driver = driverId;

const donation = delivery.donation;

await delivery.save();

// Update driver status to BUSY

await User.findByIdAndUpdate(driverId, { driverStatus: "BUSY" });

// Update status of delivery to On Arrive

await Donation.findByIdAndUpdate(donation, { status: "On Arrive" });

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Delivery successfully claimed",

data: { delivery }

});

});

// Update delivery status (CLAIMED → IN\_TRANSIT → DELIVERED)

const updateDeliveryStatus = asyncWrapper(async (req, res, next) => {

const driverId = req.currentUser.id;

const { id } = req.params;

const { status } = req.body;

if (!status || !["IN\_TRANSIT", "DELIVERED", "CANCELED"].includes(status)) {

return next(appError.create("Invalid or missing status", 400, httpStatusText.FAIL));

}

const delivery = await DeliveryRequest.findById(id);

if (!delivery) {

return next(appError.create("Delivery request not found", 404, httpStatusText.FAIL));

}

if (!delivery.driver || delivery.driver.toString() !== driverId) {

return next(appError.create("You are not assigned to this delivery", 403, httpStatusText.FAIL));

}

if (["DELIVERED", "CANCELED"].includes(delivery.status)) {

return next(appError.create("This delivery is already completed or canceled", 400, httpStatusText.FAIL));

}

// Rules for allowed transitions

if (status === "IN\_TRANSIT" && delivery.status !== "CLAIMED") {

return next(appError.create("Only claimed deliveries can be marked as in transit", 400, httpStatusText.FAIL));

}

if (status === "DELIVERED" && delivery.status !== "IN\_TRANSIT") {

return next(appError.create("Only in-transit deliveries can be marked as delivered", 400, httpStatusText.FAIL));

}

delivery.status = status;

await delivery.save();

// If final status, mark driver as available

if (["DELIVERED", "CANCELED"].includes(status)) {

await User.findByIdAndUpdate(driverId, { driverStatus: "AVAILABLE" });

// if status is delivered update the donation status

if (status === "DELIVERED") {

await Donation.findByIdAndUpdate(delivery.donation, { status: "Completed" });

}else {

await Donation.findByIdAndUpdate(delivery.donation, { status: "Pending" });

}

}

res.status(200).json({

status: httpStatusText.SUCCESS,

message: `Delivery status updated to ${status}`,

data: { delivery }

});

});

// Get all delivery requests for current user (donor or driver)

const getMyDeliveryRequests = asyncWrapper(async (req, res) => {

const userId = req.currentUser.id;

const role = req.currentUser.role;

console.log("User ID:", userId);

console.log("User Role:", role);

const query = role === "DONOR"

? { donor: userId }

: role === "DRIVER"

? { driver: userId }

: {};

const deliveries = await DeliveryRequest.find(query)

.populate("donation", "category donationType status books clothes food material")

.populate("orphanage", "name contact location")

.populate("driver", "firstName lastName email")

.populate("donor", "firstName lastName email");

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { deliveries }

});

});

// Get single delivery request by ID

const getDeliveryRequestById = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const delivery = await DeliveryRequest.findById(id)

.populate("donation", "category donationType status books clothes food material")

.populate("orphanage", "name contact location")

.populate("driver", "firstName lastName email")

.populate("donor", "firstName lastName email");

if (!delivery) {

return next(appError.create("Delivery request not found", 404, httpStatusText.FAIL));

}

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { delivery }

});

});

// Driver updates current location

const updateDriverLocation = asyncWrapper(async (req, res, next) => {

const driverId = req.currentUser.id;

const { latitude, longitude } = req.body;

// of if its a number in string format

if (!latitude || !longitude) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

if (isNaN(latitude) || isNaN(longitude)) {

return next(appError.create("Coordinates must be numbers", 400, httpStatusText.FAIL));

}

const updated = await User.findByIdAndUpdate(

driverId,

{

driverCurrentLocation: {

type: "Point",

coordinates: [latitude, longitude],

},

},

{ new: true }

);

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Location updated",

data: { location: updated.driverCurrentLocation },

});

});

// Get current driver location for a delivery

const getDriverLocation = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const delivery = await DeliveryRequest.findById(id).populate("driver");

if (!delivery) {

return next(appError.create("Delivery request not found", 404, httpStatusText.FAIL));

}

if (!delivery.driver) {

return next(appError.create("No driver assigned yet", 400, httpStatusText.FAIL));

}

const location = delivery.driver.driverCurrentLocation;

if (!location) {

return next(appError.create("Driver location not found", 404, httpStatusText.FAIL));

}

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { driverLocation: location },

});

}

);

// get all busy drivers locations

const getAllBusyDriversLocations = asyncWrapper(async (req, res, next) => {

const busyDrivers = await User.find({ role: userRoles.DRIVER, driverStatus: "BUSY" });

if (!busyDrivers.length) {

return next(appError.create("No busy drivers found", 404, httpStatusText.FAIL));

}

const locations = busyDrivers.map(driver => ({

id: driver.\_id,

location: driver.driverCurrentLocation

}));

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { locations }

});

});

// driver route map link generator

const generateDriverRouteLink = asyncWrapper(async (req, res, next) => {

const { deliveryId } = req.body;

const driverId = req.currentUser.id;

if (!deliveryId) {

return next(appError.create("Missing delivery ID", 400, httpStatusText.FAIL));

}

const delivery = await DeliveryRequest.findById(new mongoose.Types.ObjectId(deliveryId));

if (!delivery) {

return next(appError.create("Delivery request not found", 404, httpStatusText.FAIL));

}

if (!delivery.driver) {

return next(appError.create("No driver assigned yet", 400, httpStatusText.FAIL));

}

if(delivery.driver.toString() !== driverId.toString()) {

return next(appError.create("You are not assigned to this delivery", 403, httpStatusText.FAIL));

}

const mapLink = `${process.env.FRONTEND\_URL}/driver-route.html?deliveryId=${deliveryId}`;

if (!mapLink) {

return next(appError.create("Failed to generate map link", 500, httpStatusText.FAIL));

}

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Driver route link generated",

data: { mapLink }

});

});

module.exports = {

createDeliveryRequest,

claimDeliveryRequest,

updateDeliveryStatus,

getMyDeliveryRequests,

getDeliveryRequestById,

updateDriverLocation,

getDriverLocation,

getAllBusyDriversLocations,

generateDriverRouteLink

};

**File: deliveryRequest.model.js**

const mongoose = require("mongoose");

const deliveryRequestSchema = new mongoose.Schema({

donor: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

donation: {type: mongoose.Schema.Types.ObjectId, ref: "Donation", required: true},

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage" },

status: {

type: String,

enum: ["PENDING", "CLAIMED", "IN\_TRANSIT", "DELIVERED", "CANCELED"],

default: "PENDING"

},

location: {

type: {

type: String,

enum: ["Point"],

default: "Point"

},

coordinates: {

type: [Number],

required: true

}

},

destination: {

type: {

type: String,

enum: ["Point"],

default: "Point"

},

coordinates: {

type: [Number],

required: true

}

},

driver: { type: mongoose.Schema.Types.ObjectId, ref: "User", default: null },

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

});

// Create a 2dsphere index for geospatial queries

// This allows us to perform geospatial queries on the location and destination fields

// The 2dsphere index is used for queries that involve spherical geometry

deliveryRequestSchema.index({ location: "2dsphere" });

deliveryRequestSchema.index({ destination: "2dsphere" });

module.exports = mongoose.model("DeliveryRequest", deliveryRequestSchema);

**File: deliveryRequest.route.js**

const express = require("express");

const router = express.Router();

const deliveryController = require("../controllers/deliveryRequest.controller.js");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

// DONOR: Create new delivery request

router.post(

"/",

verifyToken,

allowedTo(userRoles.DONOR),

deliveryController.createDeliveryRequest

);

// DRIVER: Claim delivery request

router.post(

"/claim/:id",

verifyToken,

allowedTo(userRoles.DRIVER),

deliveryController.claimDeliveryRequest

);

// driver route map link generator

router.post(

"/driver-route",

verifyToken,

allowedTo(userRoles.DRIVER),

deliveryController.generateDriverRouteLink

);

// DRIVER: Update delivery status (CLAIMED → IN\_TRANSIT → DELIVERED)

router.patch(

"/:id/status",

verifyToken,

allowedTo(userRoles.DRIVER),

deliveryController.updateDeliveryStatus

);

// Get all delivery requests for current user (donor or driver)

router.get(

"/my",

verifyToken,

allowedTo(userRoles.DONOR, userRoles.DRIVER),

deliveryController.getMyDeliveryRequests

);

// Get delivery request by ID

router.get(

"/:id",

verifyToken,

deliveryController.getDeliveryRequestById

);

// \*\*\*\*\*\*\*\*\*\*\*\*\*

// Driver updates current location

router.put(

"/drivers/location",

verifyToken,

allowedTo(userRoles.DRIVER),

deliveryController.updateDriverLocation

);

// \*\*\*\*\*\*\*\*\*\*\*\*\*

// Get current driver location for a delivery

router.get(

"/:id/track",

verifyToken,

deliveryController.getDriverLocation

);

// Get all busy drivers locations

router.get(

"/drivers/busy/locations",

verifyToken,

deliveryController.getAllBusyDriversLocations

);

module.exports = router;

**File: donation.controller.js**

const Donation = require("../models/donation.model.js");

const User = require("../models/user.model.js");

const Orphan = require("../models/orphan.model.js");

const Orphanage = require("../models/orphanage.model.js");

const asyncWrapper = require("../middlewares/asyncWrapper.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const SystemSettings = require("../models/systemSettings.model.js");

const ExcelJS = require("exceljs");

const stripe = require("stripe")(process.env.STRIPE\_SECRET\_KEY);

const userRoles = require('../utilities/userRoles');

const calculateFees = async (amount) => {

const settings = await SystemSettings.findOne();

const feePercent = settings ? settings.transactionFeePercent : 5; // 5% كافتراضي

const fee = Math.round(amount \* (feePercent / 100));

const net = amount - fee;

return { fee, net };

};

const createBooksDonation = asyncWrapper(async (req, res, next) => {

const { books, orphanage, supportProgram } = req.body;

const donor = req.currentUser.id;

if (!books ) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const donation = new Donation({

donor,

category: "Education Support",

donationType: "Books",

books,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Books donation created successfully",

data: { donation }

});

});

//////// creat midical aid material

const createMidicalMaterial = asyncWrapper(async (req, res, next) => {

const { material, orphanage,supportProgram } = req.body;

const donor = req.currentUser.id;

if (!material ) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const donation = new Donation({

donor,

category: "Medical Aid",

donationType: "Material",

material,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Medical Aid Material donation created successfully",

data: { donation }

});

});

/////// creat Education material

const createEducationMaterial = asyncWrapper(async (req, res, next) => {

const {material, orphanage, supportProgram } = req.body;

const donor = req.currentUser.id;

if (!material ) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const donation = new Donation({

donor,

category: "Education Support",

donationType: "Material",

material,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Education Support Material donation created successfully",

data: { donation }

});

});

/////// creat General Food

const createGeneralFood = asyncWrapper(async (req, res, next) => {

const {food, orphanage, supportProgram } = req.body;

const donor = req.currentUser.id;

if (!food ) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const donation = new Donation({

donor,

category: "General Fund",

donationType: "Food",

food,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "General Fund Food donation created successfully",

data: { donation }

});

});

/////// creat General Clothes

const createGeneralClothes = asyncWrapper(async (req, res, next) => {

const {clothes, orphanage, supportProgram } = req.body;

const donor = req.currentUser.id;

if (!clothes ) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const donation = new Donation({

donor,

category: "General Fund",

donationType: "Clothes",

clothes,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "General Fund Clothes donation created successfully",

data: { donation }

});

});

const createDonationEducattionFinancial = asyncWrapper(async (req, res, next) => {

const { amount, orphanage, supportProgram} = req.body;

const donor = req.currentUser.id;

const category="Education Support";

// Basic validation

if (!donor || !amount) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

// احسب الخصم والمبلغ الصافي

const { fee, net } = await calculateFees(amount);

const donation = new Donation({

donor,

category,

donationType: "Financial",

amount,

fee,

netAmount: net,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

const session = await stripe.checkout.sessions.create({

payment\_method\_types: ["card"],

mode: "payment",

line\_items: [

{

price\_data: {

currency: "usd",

product\_data: {

name: `Donation - ${category}`,

},

unit\_amount: amount \* 100,

},

quantity: 1,

},

],

metadata: {

donationId: donation.\_id.toString(),

fee: fee.toString(),

netAmount: net.toString(),

},

success\_url: `${process.env.DOMAIN}/success`,

cancel\_url: `${process.env.DOMAIN}/cancel`,

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

donation.transactionId = session.payment\_intent || session.id;

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Donation session created successfully",

data: {

donationId: donation.\_id,

checkoutUrl: session.url,

},

});

});

/////general-financial

const createDonationGeneralFinancial = asyncWrapper(async (req, res, next) => {

const { amount, orphanage, supportProgram} = req.body;

const donor = req.currentUser.id;

const category="General Fund";

// Basic validation

if (!donor || !amount) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const { fee, net } = await calculateFees(amount);

const donation = new Donation({

donor,

category,

donationType: "Financial",

amount,

fee,

netAmount: net,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

// Stripe Checkout

const session = await stripe.checkout.sessions.create({

payment\_method\_types: ["card"],

mode: "payment",

line\_items: [

{

price\_data: {

currency: "usd",

product\_data: {

name: `Donation - ${category}`,

},

unit\_amount: amount \* 100,

},

quantity: 1,

},

],

metadata: {

donationId: donation.\_id.toString(),

fee: fee.toString(),

netAmount: net.toString(),

},

success\_url: `${process.env.DOMAIN}/success`,

cancel\_url: `${process.env.DOMAIN}/cancel`,

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

donation.transactionId = session.payment\_intent || session.id;

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Donation session created successfully",

data: {

donationId: donation.\_id,

checkoutUrl: session.url,

},

});

});

/////general-financial

const createDonationMidicalFinancial = asyncWrapper(async (req, res, next) => {

const { amount, orphanage, supportProgram} = req.body;

const donor = req.currentUser.id;

const category="Medical Aid";

// Basic validation

if (!donor || !amount) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

// ✅ احسب الخصم والمبلغ الصافي

const { fee, net } = await calculateFees(amount);

const donation = new Donation({

donor,

category,

donationType: "Financial",

amount,

fee,

netAmount: net,

orphanage,

supportProgram,

status: "Pending",

transactionId: "TEMP"

});

const session = await stripe.checkout.sessions.create({

payment\_method\_types: ["card"],

mode: "payment",

line\_items: [

{

price\_data: {

currency: "usd",

product\_data: {

name: `Donation - ${category}`,

},

unit\_amount: amount \* 100,

},

quantity: 1,

},

],

metadata: {

donationId: donation.\_id.toString(),

fee: fee.toString(),

netAmount: net.toString(),

},

success\_url: `${process.env.DOMAIN}/success`,

cancel\_url: `${process.env.DOMAIN}/cancel`,

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

donation.transactionId = session.payment\_intent || session.id;

await donation.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Donation session created successfully",

data: {

donationId: donation.\_id,

checkoutUrl: session.url,

},

});

});

//////get all donations

const getAllDonations = asyncWrapper(async (req, res, next) => {

const donations = await Donation.find()

.populate("donor", "firstName lastName role \_id")

.populate("orphanage", "name location")

.populate("supportProgram", "name description"); // إذا كنت بحاجة لعرض الدعم الخاص بالتبرع.

if (!donations) {

return next(appError.create("No donations found", 404, httpStatusText.FAIL));

}

return res.json({ status: httpStatusText.SUCCESS, data: { donations } });

});

//////////////get donation by ID

const getDonationsByOrphanage = asyncWrapper(async (req, res, next) => {

const { orphanageid } = req.params;

const currentUser = req.currentUser;

// Check if user is orphanage admin

if (currentUser.role === userRoles.ORPHANAGE\_ADMIN) {

const orphanage = await Orphanage.findOne({ \_id: orphanageid, admin: currentUser.id });

if (!orphanage) {

return next(appError.create("Unauthorized to access donations for this orphanage", 403, httpStatusText.FAIL));

}

}

// Admin can view any, donor not allowed here (unless you want to handle that too)

const donations = await Donation.find({ orphanage: orphanageid })

.populate("donor", "firstName lastName role \_id")

.populate("orphanage", "name location")

.populate("supportProgram", "name description");

if (!donations || donations.length === 0) {

return next(appError.create("No donations found for this orphanage", 404, httpStatusText.FAIL));

}

return res.json({ status: httpStatusText.SUCCESS, data: { donations } });

});

const getDonationDonor = asyncWrapper(async (req, res, next) => {

const donor = req.currentUser.id; // Get the current logged-in user (donor)

// Find donations associated with the donor

const donations = await Donation.find({ donor: donor })

.populate("donor", "firstName lastName role \_id") // populate added for 'createdBy' to get 'role' and '\_id'

.populate("supportProgram", "name description")

.populate("orphanage", "name location");

if (!donations || donations.length === 0) {

return next(appError.create("No donations found for this donor", 404, httpStatusText.FAIL));

}

return res.json({

status: httpStatusText.SUCCESS,

message: "Donations retrieved successfully",

data: { donations },

});

});

// Get a single donation by ID

const getDonationById = asyncWrapper(async (req, res, next) => {

const { id } = req.params; // استخراج الـ id من الـ params

// البحث عن التبرع بناءً على الـ id

const donation = await Donation.findById(id).populate("orphanage", "name location");

if (!donation) {

return next(appError.create("Donation not found", 404, httpStatusText.FAIL));

}

return res.json({

status: httpStatusText.SUCCESS,

message: "Donation retrieved successfully",

data: { donation },

});

});

const exportFeesReportExcel = asyncWrapper(async (req, res, next) => {

const { startDate, endDate } = req.query;

const filter = {

donationType: "Financial",

status: "Completed"

};

if (startDate || endDate) {

filter.createdAt = {};

if (startDate) filter.createdAt.$gte = new Date(startDate);

if (endDate) filter.createdAt.$lte = new Date(endDate);

}

const donations = await Donation.find(filter)

.populate("donor", "firstName lastName email")

.populate("orphanage", "name");

const settings = await SystemSettings.findOne();

const feePercentage = settings?.transactionFeePercent || 0;

const workbook = new ExcelJS.Workbook();

const worksheet = workbook.addWorksheet("Fees Report");

worksheet.columns = [

{ header: "Donor Name", key: "donor", width: 25 },

{ header: "Donor Email", key: "email", width: 30 },

{ header: "Orphanage", key: "orphanage", width: 25 },

{ header: "Category", key: "category", width: 20 },

{ header: "Donation Status", key: "status", width: 15 },

{ header: "Amount ($)", key: "amount", width: 15 },

{ header: "Fee ($)", key: "fee", width: 15 },

{ header: "Net Amount ($)", key: "net", width: 15 },

{ header: "Transaction ID", key: "transactionId", width: 30 },

{ header: "Donation Date", key: "date", width: 20 }

];

donations.forEach(d => {

const fullName = `${d.donor?.firstName || ""} ${d.donor?.lastName || ""}`.trim() || "Unknown";

worksheet.addRow({

donor: fullName,

email: d.donor?.email || "N/A",

orphanage: d.orphanage?.name || "N/A",

category: d.category || "N/A",

status: d.status,

amount: d.amount || 0,

fee: d.fee || 0,

net: d.netAmount || 0,

transactionId: d.transactionId || "N/A",

date: d.createdAt.toLocaleString()

});

});

const totalAmount = donations.reduce((sum, d) => sum + (d.amount || 0), 0);

const totalFees = donations.reduce((sum, d) => sum + (d.fee || 0), 0);

const totalNet = donations.reduce((sum, d) => sum + (d.netAmount || 0), 0);

worksheet.addRow([]);

worksheet.addRow(["Total", "", "", "", "", totalAmount, totalFees, totalNet, "", ""]);

worksheet.addRow(["Fee %", "", "", "", "", "", "", `${feePercentage}%`, "", ""]);

res.setHeader("Content-Type", "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet");

res.setHeader("Content-Disposition", `attachment; filename=fees-report-${Date.now()}.xlsx`);

await workbook.xlsx.write(res);

res.end();

});

const getDonationFinanceSummary = asyncWrapper(async (req, res, next) => {

const settings = await SystemSettings.findOne();

const feePercentage = settings?.transactionFeePercent || 0;

const financialDonations = await Donation.find({ donationType: "Financial",

status: "Completed"

});

const totalDonations = financialDonations.reduce((sum, d) => sum + (d.amount || 0), 0);

const totalFees = financialDonations.reduce((sum, d) => sum + (d.fee || 0), 0);

const totalNetAmount = financialDonations.reduce((sum, d) => sum + (d.netAmount || 0), 0);

const totalTransactions = financialDonations.length;

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Financial donation summary retrieved",

data: {

totalTransactions,

totalDonations,

totalFees,

totalNetAmount,

feePercentage: `${feePercentage}%`

}

});

});

module.exports = {

createDonationEducattionFinancial,//lama

getAllDonations,

getDonationsByOrphanage,

getDonationDonor,

getDonationById,

createBooksDonation,

createDonationGeneralFinancial,//lama

createDonationMidicalFinancial,//lama

createMidicalMaterial,

createEducationMaterial,

createGeneralFood,

createGeneralClothes,

exportFeesReportExcel,//lama

getDonationFinanceSummary//lama

};

**File: donation.model.js**

// const mongoose = require('mongoose');

// const DonationSchema = new mongoose.Schema({

// donor: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

// category: {

// type: String,

// enum: ["General Fund", "Education Support", "Medical Aid", "Emergency Relief"],

// required: true

// },

// amount: { type: Number, required: true },

// transactionId: { type: String, required: true },

// status: { type: String, enum: ["Pending", "Completed"], default: "Pending" },

// orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage", required: false },

// createdAt: { type: Date, default: Date.now }

// });

// module.exports = mongoose.model("Donation", DonationSchema);

const mongoose = require('mongoose');

const DonationSchema = new mongoose.Schema({

fee: { type: Number, default: 0 },//نسبة الخصم

netAmount: { type: Number,

default: function () {

return this.amount;

}

},

donor: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

category: {

type: String,

enum: ["General Fund", "Education Support", "Medical Aid", "Emergency Relief"],

required: true

},

donationType: {

type: String,

enum: ["Books", "Clothes", "Food", "Financial", "Material"],

required: true

},

amount: { type: Number, default: 0 }, // المبلغ (في حالة التبرعات المالية)

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

status: { type: String, enum: ["Pending", "Completed" , "On Arrive", "Controlled"], default: "Pending" },

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage", required: false },

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

supportProgram: { type: mongoose.Schema.Types.ObjectId, ref: "SupportProgram"}, // إضافة هذا الحقل

books: [{

name: { type: String },

quantity: { type: Number }

}],

clothes: [{

type: { type: String }, // نوع الملابس (مثل T-shirt، jacket، إلخ)

size: { type: String }, // الحجم (مثل S, M, L)

quantity: { type: Number } // الكمية

}],

food: [{

type: { type: String },

quantity: { type: Number }

}],

material: [{

type: { type: String },

quantity: { type: Number }

}],

campaign: { type: mongoose.Schema.Types.ObjectId, ref: "EmergencyCampaign", default: null, required: false }

});

module.exports = mongoose.model("Donation", DonationSchema);

**File: donation.route.js**

const express = require("express");

const router = express.Router();

const donationController = require("../controllers/donation.controller.js");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

// creat donation educational support - financial

router.post( "/educational-support/financial",verifyToken,allowedTo( userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN),donationController.createDonationEducattionFinancial);

// creat donation general - financial

router.post( "/genaral-fund/financial",verifyToken,allowedTo( userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN),donationController.createDonationGeneralFinancial);

// Create donation for medical aid financial

router.post("/medical-aid/financial", verifyToken, allowedTo(userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN), donationController.createDonationMidicalFinancial);

// Create donation for educational support - books

router.post("/educational-support/books", verifyToken, allowedTo(userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN), donationController.createBooksDonation);

// Create donation for medical aid - material

router.post("/medical-aid/material", verifyToken, allowedTo(userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN), donationController.createMidicalMaterial);

// Create donation for educational - material

router.post("/educational-support/material", verifyToken, allowedTo(userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN), donationController.createEducationMaterial);

// creat donation general - food

router.post( "/genaral-fund/food",verifyToken,allowedTo( userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN),donationController.createGeneralFood);

// creat donation general - clothes

router.post( "/genaral-fund/clothes",verifyToken,allowedTo( userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN),donationController.createGeneralClothes);

// get donation

router.get("/", verifyToken, allowedTo(userRoles.ADMIN, userRoles.SUPPORT\_PROGRAM\_ADMIN), donationController.getAllDonations);

// get donation by id

router.get("/orphanage/:orphanageid", verifyToken, allowedTo(userRoles.ADMIN, userRoles.ORPHANAGE\_ADMIN), donationController.getDonationsByOrphanage);//check

// Get donations for the logged-in donor

router.get("/mine", verifyToken, allowedTo(userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN), donationController.getDonationDonor);

// Get a single donation by ID

router.get("/:id", verifyToken, allowedTo(userRoles.DONOR, userRoles.ADMIN, userRoles.SUPPORT\_PROGRAM\_ADMIN), donationController.getDonationById);

router.get(

"/admin/export-fees",

verifyToken,

allowedTo(userRoles.ADMIN),

donationController.exportFeesReportExcel

);

router.get(

"/admin/financial-summary",

verifyToken,

allowedTo(userRoles.ADMIN),

donationController.getDonationFinanceSummary

);

module.exports = router;

**File: emergencyCampaign.controller.js**

const mongoose = require("mongoose");

const EmergencyCampaign = require("../models/emergencyCampaign.model.js");

const asyncWrapper = require("../middlewares/asyncWrapper.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const stripe = require("stripe")(process.env.STRIPE\_SECRET\_KEY);

const Donation = require("../models/donation.model");

const Orphanage = require("../models/orphanage.model.js");

const userRoles = require("../utilities/userRoles.js");

const User = require("../models/user.model");

const sendEmail = require("../utilities/sendEmail");

const donateToCampaign = asyncWrapper(async (req, res, next) => {

try {

const { id: campaignId } = req.params;

const { amount, orphanage } = req.body;

const donor = req.currentUser.id;

// 0. Validate orphanage

const orphanageFound = await Orphanage.findById(orphanage);

if (!orphanageFound) {

return next(appError.create("Orphanage not found", 404, httpStatusText.ERROR));

}

// check if "orphanage" is the creator of this "campaign"

// 1. Validate campaign

const campaign = await EmergencyCampaign.findById(campaignId);

if (!campaign || campaign.status !== "Active") {

return next(appError.create("Invalid or inactive campaign" , 400, httpStatusText.FAIL));

}

else{

if(orphanage !== campaign.orphanage.toString()){

return next(appError.create("The entered orphanage is not the creator of this campaign." , 400, httpStatusText.FAIL));

}

}

if (new Date(campaign.endDate) < new Date()) {

return next(appError.create("Campaign has already ended.", 400, httpStatusText.FAIL));

}

// 2. Validate amount

if (!amount || amount <= 0) {

return next(appError.create("Invalid amount. Please enter a positive number", 400, httpStatusText.FAIL));

}

// 2. Create Donation (Pending)

const donation = new Donation({

donor,

category: "Emergency Relief",

donationType: "Financial",

amount,

transactionId: "TEMP",

status: "Pending",

campaign: campaignId

});

await donation.save();

// 3. Create Stripe Checkout Session

const session = await stripe.checkout.sessions.create({

payment\_method\_types: ["card"],

mode: "payment",

line\_items: [

{

price\_data: {

currency: "usd",

product\_data: {

name: `Emergency Campaign Donation`,

},

unit\_amount: amount \* 100,

},

quantity: 1,

},

],

metadata: {

donationId: donation.\_id.toString(),

},

success\_url: `${process.env.DOMAIN}/success`,

cancel\_url: `${process.env.DOMAIN}/cancel`,

});

await Orphanage.findByIdAndUpdate(orphanage, { $push: { donations: donation.\_id } });

// Update transaction ID temporarily with Stripe session ID

donation.transactionId = session.payment\_intent || session.id;

await donation.save();

return res.status(201).json({

message: "Donation initialized. Complete payment using the URL.",

url: session.url,

});

} catch (error) {

console.error("💥 Donation Error:", error);

res.status(500).json({ message: "Failed to process donation." });

}

}

);

// make sure to keep traking the end date and the status (if reaches the end date without reaching target amount then change to expired) of the created campaign

// Create a new campaign

const createCampaign = asyncWrapper(async (req, res, next) => {

const { title, description, targetAmount, endDate, orphanage } = req.body;

const user = req.currentUser;

const orphanageDoc = await Orphanage.find({admin: new mongoose.Types.ObjectId(user.id)});

if(user.role !== userRoles.ADMIN){

if (!orphanageDoc[0]) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

}

if (

user.role !== userRoles.ADMIN &&

(user.role !== userRoles.ORPHANAGE\_ADMIN || orphanageDoc[0].\_id.toString() !== orphanage)

) {

return next(appError.create("Unauthorized action or Orphanage Admin mismatch", 403, httpStatusText.FAIL));

}

// Validate campaign

if (!title || !description || !targetAmount || !endDate) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

const campaign = await EmergencyCampaign.create({

title,

description,

targetAmount,

endDate,

orphanage

});

// after campaign is created

const donors = await User.find({ role: "DONOR" });

const html = `

<h2>🚨 New Emergency Campaign Launched!</h2>

<p><strong>${campaign.title}</strong></p>

<p>${campaign.description}</p>

<p>Target: $${campaign.targetAmount}</p>

<p><a href="${process.env.FRONTEND\_URL}/campaigns/${campaign.\_id}">View Campaign</a></p>

`;

for (const donor of donors) {

await sendEmail({

to: donor.email,

subject: "🚨 New Emergency Campaign Needs Your Help",

html,

});

}

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Emergency campaign created successfully",

data: { campaign }

});

});

// Get all campaigns (optionally filter by status)

const getCampaigns = asyncWrapper(async (req, res) => {

const { status } = req.query;

const filter = {};

if (status) filter.status = status;

const campaigns = await EmergencyCampaign.find(filter).populate("orphanage", "name location contact");

if (!campaigns || campaigns.length === 0) {

return res.status(404).json({

status: httpStatusText.FAIL,

message: "No campaigns found"

});

}

// also find donations for each campaign

const donations = await Donation.find({ campaign: campaigns.map(c => c.\_id) })

.select("category donationType amount status createdAt campaign")

.populate("donor", "name email");

const donationsByCampaign = {};

donations.forEach(donation => {

if (!donationsByCampaign[donation.campaign]) {

donationsByCampaign[donation.campaign] = [];

}

donationsByCampaign[donation.campaign].push(donation);

});

// now send each campaign with its donations (immediately in response)

const campaignsWithDonations = campaigns.map(campaign => {

return {

...campaign.\_doc,

donations: donationsByCampaign[campaign.\_id] || []

};

});

// send the campaigns with their donations

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { campaigns: campaignsWithDonations }

});

});

// Get one campaign by ID

const getCampaignById = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const campaign = await EmergencyCampaign.findById(id).populate("orphanage", "name location contact");

if (!campaign) {

return next(appError.create("Campaign not found", 404, httpStatusText.FAIL));

}

// also find donations for this campaign

const donations = await Donation.find({ campaign: id })

.select("category donationType amount status createdAt campaign")

.populate("donor", "name email");

const donationsByCampaign = {};

donations.forEach(donation => {

if (!donationsByCampaign[donation.campaign]) {

donationsByCampaign[donation.campaign] = [];

}

donationsByCampaign[donation.campaign].push(donation);

});

// now send the campaign with its donations (immediately in response)

const campaignWithDonations = {

...campaign.\_doc,

donations: donationsByCampaign[campaign.\_id] || []

};

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { campaignWithDonations }

});

});

// Get all donations for a specific campaign

const getCampaignDonations = asyncWrapper(async (req, res, next) => {

const { id: campaignId } = req.params;

const user = req.currentUser;

const campaign = await EmergencyCampaign.findById(campaignId);

if (!campaign) {

return next(appError.create("Campaign not found", 404, httpStatusText.FAIL));

}

const orphanage = await Orphanage.find({admin: new mongoose.Types.ObjectId(user.id)});

if(user.role !== userRoles.ADMIN){

if (!orphanage[0]) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

}

// Check access permissions

if (

user.role !== userRoles.ADMIN &&

(user.role !== userRoles.ORPHANAGE\_ADMIN || orphanage[0].\_id.toString() !== campaign.orphanage.toString())

) {

return next(appError.create("Unauthorized access or Orphanage Admin mismatch", 403, httpStatusText.FAIL));

}

const donations = await Donation.find({ campaign: campaignId, status: "Completed" })

.select("\_id donor category donationType amount status createdAt")

.populate("donor", "name email");

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { donations }

});

});

// Get campaign summary (for admin and orphanage admin)

const getCampaignSummary = asyncWrapper(async (req, res, next) => {

const campaignId = req.params.id;

const user = req.currentUser;

const campaign = await EmergencyCampaign.findById(campaignId).populate("orphanage");

if (!campaign) {

return next(appError.create("Campaign not found", 404, httpStatusText.FAIL));

}

const orphanage = await Orphanage.find({admin: new mongoose.Types.ObjectId(user.id)});

if(user.role !== userRoles.ADMIN){

if (!orphanage[0]) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

}

// Authorization (admin or owning orphanage)

if (user.role !== userRoles.ADMIN &&

(user.role !== userRoles.ORPHANAGE\_ADMIN || orphanage[0].\_id.toString() !== campaign.orphanage.\_id.toString())) {

return next(appError.create("Not authorized to view this campaign summary or Orphanage Admin mismatch", 403, httpStatusText.FAIL));

}

// Get all donations for this campaign

const donations = await Donation.find({ campaign: campaignId, status: "Completed" });

const totalDonations = donations.reduce((sum, d) => sum + d.amount, 0);

const donorIds = new Set(donations.map(d => d.donor.toString()));

console.log("🎯 Donor IDs:", donations.map(d => d.donor.toString()));

const donorCount = donorIds.size;

const progressPercent = (campaign.raisedAmount / campaign.targetAmount) \* 100;

const today = new Date();

const endDate = new Date(campaign.endDate);

const timeDiff = Math.max(endDate - today, 0);

const daysRemaining = Math.ceil(timeDiff / (1000 \* 60 \* 60 \* 24));

res.status(200).json({

status: httpStatusText.SUCCESS,

data: {

campaign: {

title: campaign.title,

targetAmount: campaign.targetAmount,

currentAmount: campaign.currentAmount,

status: campaign.status,

endDate: campaign.endDate

},

totalDonations,

donorCount,

progressPercent: Math.round(progressPercent),

daysRemaining

}

});

});

// Update a campaign (only for admin or the owning orphanage)

const updateCampaign = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const { title, description, targetAmount, endDate } = req.body;

const user = req.currentUser;

const campaign = await EmergencyCampaign.findById(id);

if (!campaign) {

return next(appError.create("Campaign not found", 404, httpStatusText.FAIL));

}

const orphanage = await Orphanage.find({admin: new mongoose.Types.ObjectId(user.id)});

if(user.role !== userRoles.ADMIN){

if (!orphanage[0]) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

}

// Only allow updates for active campaigns

if (campaign.status !== "Active") {

return next(appError.create("Only active campaigns can be updated", 400, httpStatusText.FAIL));

}

// Allow only admin or the owning orphanage

if (

user.role !== userRoles.ADMIN &&

(!campaign.orphanage || campaign.orphanage.toString() !== orphanage[0].\_id.toString())

) {

return next(appError.create("Not authorized to update this campaign or Orphanage Admin mismatch", 403, httpStatusText.FAIL));

}

// Update allowed fields

if (title) campaign.title = title;

if (description) campaign.description = description;

if (targetAmount) campaign.targetAmount = targetAmount;

if (endDate) campaign.endDate = endDate;

await campaign.save();

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Campaign updated successfully",

data: { campaign }

});

});

// Delete a campaign (only for admin or the owning orphanage)

const deleteCampaign = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const user = req.currentUser;

const campaign = await EmergencyCampaign.findById(id);

if (!campaign) {

return next(appError.create("Campaign not found", 404, httpStatusText.FAIL));

}

const orphanage = await Orphanage.find({admin: new mongoose.Types.ObjectId(user.id)});

if(user.role !== userRoles.ADMIN){

if (!orphanage[0]) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

}

// Only allow deletion if status is still active

if (campaign.status !== "Active") {

return next(appError.create("Only active campaigns can be deleted", 400, httpStatusText.FAIL));

}

// Authorization: Only admin or campaign's orphanage

if (

user.role !== userRoles.ADMIN &&

(!campaign.orphanage || campaign.orphanage.toString() !== orphanage[0].\_id.toString())

) {

return next(appError.create("Not authorized to delete this campaign or Orphanage Admin mismatch", 403, httpStatusText.FAIL));

}

await EmergencyCampaign.findByIdAndDelete(id);

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Campaign deleted successfully"

});

});

module.exports = {

donateToCampaign,

createCampaign,

getCampaigns,

getCampaignById,

getCampaignDonations,

getCampaignSummary,

updateCampaign,

deleteCampaign

};

**File: emergencyCampaign.model.js**

const mongoose = require('mongoose');

const EmergencyCampaignSchema = new mongoose.Schema({

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

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

targetAmount: { type: Number, required: true },

raisedAmount: { type: Number, default: 0 },

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage", required: true },

status: { type: String, enum: ["Active", "Completed", "Expired"], default: "Active" },

endDate: { type: Date, required: true },

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

});

module.exports = mongoose.model("EmergencyCampaign", EmergencyCampaignSchema);

**File: emergencyCampaign.route.js**

const express = require("express");

const router = express.Router();

const campaignController = require("../controllers/emergencyCampaign.controller");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

const checkOrphanage = require("../middlewares/checkOrphanage.js");

// Basic routes

router

.route("/")

.post(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN, userRoles.ADMIN), campaignController.createCampaign)

.get(campaignController.getCampaigns);

router

.route("/:id")

.get(campaignController.getCampaignById)

.put(

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN, userRoles.ADMIN),

campaignController.updateCampaign

)

.delete(

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN, userRoles.ADMIN),

campaignController.deleteCampaign

);

router

.route("/:id/donate")

.post(verifyToken, allowedTo(userRoles.DONOR), campaignController.donateToCampaign);

router

.route("/:id/donations")

.get(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN, userRoles.ADMIN), campaignController.getCampaignDonations);

router.get("/:id/summary", verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN, userRoles.ADMIN), campaignController.getCampaignSummary);

module.exports = router;

**File: index.js**

require('dotenv').config(); // config() by default finds .env file and load it into process.env

const express = require('express');

const cors = require('cors');

const httpStatusText = require('./utilities/httpStatusText.js');

const path = require('path');

const app = express();

app.use('/uploads', express.static(path.join(\_\_dirname, 'uploads'))); // handle files (paths) invoking

const mongoose = require('mongoose');

const url = process.env.MONGO\_URL; // sensitive data

mongoose.connect(url).then(() => {

console.log('Connected to MongoDB');

});

app.use(cors());

app.use((req, res, next) => {

if (req.originalUrl === "/webhooks/stripe") {

next(); // Skip JSON parsing for Stripe webhook

} else {

express.json()(req, res, next);// tells express to handle json body comes to us

}

});

//or:

// app.use(bodyParser.json()); // you have to install the middleware from internet first

const coursesRouter = require('./routes/courses.route.js');

const usersRouter = require('./routes/users.route.js');

const orphanageRouter = require("./routes/orphanages.route.js");

const orphansRouter = require("./routes/orphans.route.js");

const sponsorshipsRouter = require("./routes/sponsorships.route.js");

const handleWebhook = require("./routes/webhook.route.js");

const campaignRoutes = require("./routes/emergencyCampaign.route.js");

const deliveryRequestRouter = require("./routes/deliveryRequest.route.js");

const donationRouter = require('./routes/donation.route.js');

const volunteerApplicationsRoutes = require("./routes/volunteerApplications.route.js");

const orphanageVolunteerRequestsRoutes = require("./routes/orphanageVolunteerRequests.route.js");

const orphanageApplicationsRoutes= require("./routes/orphanageApplications.route.js");

const reviewRoutes = require('./routes/review.route.js');

const controllingDonationRouter = require("./routes/controlDonation.route.js");

const dashboardRoutes = require("./routes/dashboard.route.js");

const settingsRoutes = require("./routes/systemSettings.route.js");

const supportProgramRoutes = require("./routes/supportProgram.route");

app.use('/uploads', express.static('uploads'));

require("./jobs/sponsorshipCompletionJob.js");

require("./jobs/campaignExpiryJob.js");

// use router as middleware

// middleware1

app.use("/webhooks", handleWebhook);

app.use('/api/courses', coursesRouter); // any request comes on '/' it will go and handle it in coursesRouter

app.use('/api/users', usersRouter);

app.use("/api/orphanages", orphanageRouter);

app.use("/api/orphans", orphansRouter);

app.use('/api/sponsorships', sponsorshipsRouter);

app.use("/api/controlling-donations", controllingDonationRouter);

app.use("/api/campaigns", campaignRoutes);

app.use('/api/donations', donationRouter);

app.use("/api/deliveryRequest", deliveryRequestRouter);

app.use("/api/orphanage/volunteer-requests", orphanageVolunteerRequestsRoutes);

app.use("/api/dashboard", dashboardRoutes);

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

//(volunteer + orphanage admin)

app.use("/api/volunteer/applications", volunteerApplicationsRoutes);

app.use("/api/orphanage/applications", orphanageApplicationsRoutes);

app.use('/api/reviews', reviewRoutes);

app.use("/api/settings", settingsRoutes);

app.use("/api/support-programs", supportProgramRoutes);

// wild card:

// middleware2 - global middleware for not found root

app.all('\*', (req, res, next) => {

return res.status(404).json({ status: httpStatusText.ERROR, message: 'Route not found' });

}); // handle all/any url passed the router without being handled

// middleware3 - global error handler

app.use((err, req, res, next) => {

return res.status(err.statusCode || 500).json({ status: err.statusText || httpStatusText.ERROR, data: null, message: /\*"Invalid Object ID"\*/ err.message, code: err.statusCode});

});

app.listen(process.env.PORT || 5000, () => {

console.log('Server is running on port 5000');

});

// express is unopinionated (does not has fixed structure - controller, middlewares, routes,... as you want).8

// while nestjs is opinionated (fixed structure - it gives me: controller, module, service, main).

// see app.route() on express website:

// app.route('/book')

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

// res.send('Get a random book')

// })

// .post((req, res) => {

// res.send('Add a book')

// })

// .put((req, res) => {

// res.send('Update the book')

// })

**File: notification.model.js**

const mongoose = require('mongoose');

const NotificationSchema = new mongoose.Schema({

user: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

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

isRead: { type: Boolean, default: false },

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

});

module.exports = mongoose.model("Notification", NotificationSchema);

**File: orphan.model.js**

const mongoose = require("mongoose");

const OrphanSchema = new mongoose.Schema({

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

age: { type: Number, required: true },

gender: { type: String, enum: ["Male", "Female"], required: true },

educationStatus: { type: String },

healthCondition: { type: String },

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage", required: true }, // Reference to orphanage

orphanageAdmin: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true }, // Assigned orphanage admin

sponsors: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }], // List of sponsors

photos: [{ type: String }],

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

});

module.exports = mongoose.model("Orphan", OrphanSchema);

**File: orphanage.model.js**

const mongoose = require("mongoose");

const requestStatus = require("../utilities/requestStatus");

const OrphanageSchema = new mongoose.Schema({

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

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

description: { type: String },

contact: {

phone: { type: String },

email: { type: String }

},

admin: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true, unique: true }, // Orphanage Admin

orphans: [{ type: mongoose.Schema.Types.ObjectId, ref: "Orphan" }], // List of orphans in this orphanage

status: { type: String, enum: [requestStatus.PENDING, requestStatus.APPROVED, requestStatus.REJECTED], default: "pending" }, // Approval status

verified: { type: Boolean, default: false }, // Will be updated by the App Admin

donations: [{ type: mongoose.Schema.Types.ObjectId, ref: "Donation" }], // List of donations for this orphanage (optional)

supportPrograms: [{ type: mongoose.Schema.Types.ObjectId, ref: "SupportProgram" }], // حقل البرامج

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

});

module.exports = mongoose.model("Orphanage", OrphanageSchema);

**File: orphanageApplications.route.js**

const express = require("express");

const router = express.Router();

const verifyToken = require("../middlewares/verifyToken");

const allowedTo = require("../middlewares/allowedTo");

const userRoles = require("../utilities/userRoles");

const controller = require("../controllers/VolunteerApplications.controller.js");

// Only ORPHANAGE\_ADMIN

// عرض كل الطلبات المقدمة لمؤسسة المشرف

router.get(

"/",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.getApplicationsForOrphanage

);

// قبول طلب تطوع

router.patch(

"/:applicationId/approve",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.approveApplication

);

// رفض طلب تطوع

router.patch(

"/:applicationId/reject",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.rejectApplication

);

module.exports = router;

**File: orphanages.controller.js**

const asyncWrapper = require('../middlewares/asyncWrapper.js');

const Orphanage = require("../models/orphanage.model.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const {validationResult} = require('express-validator');

const requestStatus = require("../utilities/requestStatus");

const Orphan = require("../models/orphan.model.js");

//get all orphanages: ok

const getAllOrphanages = asyncWrapper(

async (req, res) => {

const query = req.query;

/\* Pagination \*/

const limit = query.limit || 4;

const page = query.page || 1;

const skip = (page - 1) \* limit;

const orphanages = await Orphanage.find({}, {\_\_v: false}).limit(limit).skip(skip).populate("admin", "name email");

/\* Pagination \*/

return res.json({ status: httpStatusText.SUCCESS, data: {orphanages} });

}

);

//get orphanage by id: ok

const getOrphanageById = asyncWrapper(

async (req, res, next) => {

const id = req.params.id;

const orphanage = await Orphanage.findById(id).populate("admin", "name email");

if(!orphanage){

const error = appError.create("orphanage not found", 400, httpStatusText.FAIL);

return next(error);

}

return res.json({ status: httpStatusText.SUCCESS,data: {orphanage} });

}

);

// create new orphanage: ok

const createOrphanage = asyncWrapper(

async (req, res, next) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

const error = appError.create(errors.array(), 400, httpStatusText.FAIL);

return next(error);

}

const { name, location, description, contact: {phone, email} } = req.body;

const userId = req.currentUser.id; // User requesting the orphanage

// Check if the orphanage admin already has orphanage

const existingOrphanage = await Orphanage.findOne({

admin: userId,

status: { $in: [requestStatus.REJECTED, requestStatus.PENDING, requestStatus.APPROVED] }

});

if (existingOrphanage) {

if(existingOrphanage.status === requestStatus.REJECTED) {

return res.status(400).json({

status: httpStatusText.FAIL,

message: "You have a rejected orphanage request. You have to delete it then you can create a new one."

});

}

return next(appError.create("You already have an orphanage request pending or approved.", 400, httpStatusText.FAIL));

}

const newOrphanage = new Orphanage({

name: name,

location: location,

description: description,

contact: {

phone: phone,

email: email

},

admin: userId, // The requesting user becomes the orphanage admin

status: requestStatus.PENDING

});

await newOrphanage.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

data: {orphanage: newOrphanage},

message: "Orphanage request submitted, pending approval."

});

}

);

// approve orphanage: ok

const approveOrphanage = asyncWrapper(

async (req, res, next) => {

const orphanageId = req.params.id;

const orphanage = await Orphanage.findById(orphanageId);

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

if (orphanage.status === requestStatus.APPROVED) {

return res.json({ status: httpStatusText.SUCCESS, message: "Orphanage is already approved." });

}

orphanage.status = requestStatus.APPROVED;

orphanage.verified = true;

await orphanage.save();

res.json({

status: httpStatusText.SUCCESS,

message: "Orphanage approved successfully.",

data: { orphanage }

});

}

);

// reject orphanage: ok

const rejectOrphanage = asyncWrapper(

async (req, res, next) => {

const orphanageId = req.params.id;

const orphanage = await Orphanage.findById(orphanageId);

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

if (orphanage.status === requestStatus.REJECTED) {

return res.json({ status: httpStatusText.SUCCESS, message: "Orphanage is already rejected." });

}

orphanage.status = requestStatus.REJECTED;

orphanage.verified = true;

await orphanage.save();

res.json({

status: httpStatusText.SUCCESS,

message: "Orphanage rejected successfully.",

data: { orphanage }

});

}

);

// Update orphanage: ok

const updateOrphanage = asyncWrapper(async (req, res, next) => {

const orphanageId = req.params.id;

const updates = req.body;

const orphanage = await Orphanage.findByIdAndUpdate(orphanageId, updates, { new: true });

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

res.json({

status: httpStatusText.SUCCESS,

message: "Orphanage updated successfully.",

data: { orphanage }

});

});

// Delete orphanage: ok

const deleteOrphanage = asyncWrapper(async (req, res, next) => {

const orphangeAdmin = req.currentUser.id;

const orphanageId = req.params.id;

const orphanage = await Orphanage.findById(orphanageId);

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

if (orphanage.admin.toString() !== orphangeAdmin) {

return next(appError.create("You are not authorized to delete this orphanage", 403, httpStatusText.FAIL));

}

// delete all orphans related to this orphanage

await Orphan.deleteMany({ orphanage: orphanageId });

// delete orphanage

await Orphanage.findByIdAndDelete(orphanageId);

res.json({

status: httpStatusText.SUCCESS,

message: "Orphanage deleted successfully."

});

});

module.exports = {

getAllOrphanages,

getOrphanageById,

createOrphanage,

approveOrphanage,

rejectOrphanage,

updateOrphanage,

deleteOrphanage

}

**File: orphanages.route.js**

const express = require("express");

const router = express.Router();

const orphanageController = require("../controllers/orphanages.controller.js");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

const checkOrphanage = require("../middlewares/checkOrphanage.js");

router

.route("/")

.post(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), orphanageController.createOrphanage) // Only Orphanage Admin can create orphanage

.get(orphanageController.getAllOrphanages); // Public

router

.route("/:id")

.get(orphanageController.getOrphanageById)

.patch(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), checkOrphanage, orphanageController.updateOrphanage) // Only Orphanage Admin

.delete(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), orphanageController.deleteOrphanage); // Only Orphanage Admin

router

.route("/:id/approve")

.patch(verifyToken, allowedTo(userRoles.ADMIN), orphanageController.approveOrphanage) // Only App Admin can approve

router

.route("/:id/reject")

.patch(verifyToken, allowedTo(userRoles.ADMIN), orphanageController.rejectOrphanage); // Only App Admin can reject

module.exports = router;

**File: orphanageVolunteerRequests.route.js**

const express = require("express");

const router = express.Router();

const verifyToken = require("../middlewares/verifyToken");

const allowedTo = require("../middlewares/allowedTo");

const userRoles = require("../utilities/userRoles");

const controller = require("../controllers/VolunteerRequests.controller.js");

//Only ORPHANAGE\_ADMIN

router.get(

"/dashboard-summary",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.getOrphanageDashboardSummary

);

// إنشاء طلب تطوع جديد

router.post(

"/",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.createVolunteerRequest

);

// تعديل طلب تطوع

router.patch(

"/:id",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.updateVolunteerRequest

);

// حذف طلب تطوع

router.delete(

"/:id",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.deleteVolunteerRequest

);

// الحصول على طلب تطوع معين

router.get(

"/:id",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.getVolunteerRequestById

);

router.get(

"/",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

controller.getMyVolunteerRequests

);

module.exports = router;

**File: orphans.controller.js**

const mongoose = require("mongoose");

const Orphan = require("../models/orphan.model.js");

const Orphanage = require("../models/orphanage.model.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const { validationResult } = require("express-validator");

const asyncWrapper = require('../middlewares/asyncWrapper.js');

const requestStatus = require("../utilities/requestStatus");

// Get all orphans with pagination: ok

const getAllOrphans = asyncWrapper(

async (req, res, next) => {

const query = req.query;

const limit = query.limit || 4;

const page = query.page || 1;

const skip = (page - 1) \* limit;

const orphans = await Orphan.find({}, { \_\_v: false })

.limit(limit)

.skip(skip)

.populate("orphanage", "name location")

.populate("orphanageAdmin", "name email");

if(!orphans.length === 0){

const error = appError.create("No Orphans In Your Orphanage Yet!", 400, httpStatusText.FAIL);

return next(error);

}

return res.json({ status: httpStatusText.SUCCESS, data: { orphans } });

}

);

// Get all orphans of orphanage with pagination: ok

const getAllOrphansOfOrphanage = asyncWrapper(

async (req, res, next) => {

const query = req.query;

const limit = query.limit || 4;

const page = query.page || 1;

const skip = (page - 1) \* limit;

const orphanageId = req.params.id;

const orphanageDoc = await Orphanage.findById(new mongoose.Types.ObjectId(orphanageId));

// find orphan by it orphanage id:

const orphans = await Orphan.find({orphanage: orphanageDoc.\_id}, { \_\_v: false })

.limit(limit)

.skip(skip)

.populate("orphanage", "name location")

.populate("orphanageAdmin", "name email");

if(!orphans.length === 0){

const error = appError.create("No Orphans In Your Orphanage Yet!", 400, httpStatusText.FAIL);

return next(error);

}

return res.json({ status: httpStatusText.SUCCESS, data: { orphans } });

}

);

// Get orphan by ID: ok

const getOrphanById = asyncWrapper(

async (req, res, next) => {

const id = req.params.orphanid;

const orphan = await Orphan.findById(new mongoose.Types.ObjectId(id))

.populate("orphanage", "name location")

.populate("orphanageAdmin", "name email");

if (!orphan) {

const error = appError.create("Orphan not found", 400, httpStatusText.FAIL);

return next(error);

}

return res.json({ status: httpStatusText.SUCCESS, data: { orphan } });

}

);

const getOrphanPhotos = asyncWrapper(async (req, res, next) => {

const orphanId = req.params.orphanid;

const orphan = await Orphan.findById(orphanId);

if (!orphan) {

return next(appError.create("Orphan not found", 404, httpStatusText.FAIL));

}

const photos = orphan.photos || [];

// Build full URLs for each image

const photoUrls = photos.map(filename => `${process.env.FRONTEND\_URL}/uploads/orphans/${filename}`);

return res.status(200).json({

status: httpStatusText.SUCCESS,

data: { photos: photoUrls }

});

});

// Get orphan by ID in orphanage: ok

const getOrphanByIdInOrphanage = asyncWrapper(

async (req, res, next) => {

const orphanageId = req.params.orphanageid;

const orphanageDoc = await Orphanage.findById(new mongoose.Types.ObjectId(orphanageId));

const id = req.params.id;

const orphan = await Orphan.find({orphanage: orphanageDoc.\_id, \_id: id})

.populate("orphanage", "name location")

.populate("orphanageAdmin", "name email");

if (orphan.length === 0) {

const error = appError.create("There is No Orphan With This ID In Your Orphanage!", 400, httpStatusText.FAIL);

return next(error);

}

return res.json({ status: httpStatusText.SUCCESS, data: { orphan } });

}

);

// Create orphan: ok - {photos}

const createOrphan = asyncWrapper(

async (req, res, next) => {

const errors = validationResult(req);

if (!errors.isEmpty()) {

const error = appError.create(errors.array(), 400, httpStatusText.FAIL);

return next(error);

}

const { name, age, gender, educationStatus, healthCondition, orphanage } = req.body;

const orphanageAdmin=req.currentUser.id;

console.log(orphanageAdmin);

const orphanageDoc = await Orphanage.findById(new mongoose.Types.ObjectId(orphanage));

const photoFilenames = req.files?.map(file => file.filename) || [];

const newOrphan = new Orphan({

name:name,

age:age,

gender:gender,

educationStatus:educationStatus,

healthCondition:healthCondition,

orphanage:orphanage,

orphanageAdmin:orphanageAdmin,

photos:photoFilenames

});

await newOrphan.save();

orphanageDoc.orphans.push(newOrphan);

await orphanageDoc.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

data: { orphan: newOrphan },

message: "Orphan created successfully."

});

}

);

// Update orphan details

const updateOrphan = asyncWrapper(async (req, res, next) => {

const orphanId = req.params.orphanid;

const updates = req.body;

const orphan = await Orphan.findByIdAndUpdate(orphanId, updates, { new: true });

if (!orphan) {

return next(appError.create("Orphan not found", 404, httpStatusText.FAIL));

}

res.json({

status: httpStatusText.SUCCESS,

message: "Orphan updated successfully.",

data: { orphan }

});

});

// Delete orphan

const deleteOrphan = asyncWrapper(async (req, res, next) => {

const orphanId = req.params.orphanid;

const oldOrphan = await Orphan.findById(orphanId);

if (!oldOrphan) {

return next(appError.create("Orphan not found", 404, httpStatusText.FAIL));

}

const orphanageDoc = await Orphanage.findById(oldOrphan.orphanage);

// Delete the orphan

await Orphan.findByIdAndDelete(orphanId);

// Remove orphan from orphanage's orphans array

orphanageDoc.orphans = orphanageDoc.orphans.filter(id => !id.equals(orphanId));

// Save the updated orphanage document

await orphanageDoc.save();

res.json({

status: httpStatusText.SUCCESS,

message: "Orphan deleted successfully."

});

});

module.exports = {

getAllOrphans,

getAllOrphansOfOrphanage,

getOrphanById,

getOrphanByIdInOrphanage,

createOrphan,

updateOrphan,

deleteOrphan,

getOrphanPhotos

};

**File: orphans.route.js**

const express = require("express");

const router = express.Router();

const orphanController = require("../controllers/orphans.controller.js");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

const checkOrphanage = require("../middlewares/checkOrphanage.js");

const upload = require("../middlewares/uploadOrphanPhotos.js");

// CRUD operations for orphans

router

.route("/")

.get(orphanController.getAllOrphans) // Get all orphans

.post(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), upload.array("photos", 5), checkOrphanage, orphanController.createOrphan); // Add new orphan

router

.route("/:orphanid")

.get(orphanController.getOrphanById) // Get a single orphan by ID

.patch(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), checkOrphanage, orphanController.updateOrphan) // Update orphan details

.delete(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), checkOrphanage, orphanController.deleteOrphan); // Delete orphan

router.get("/:orphanid/photos", orphanController.getOrphanPhotos);

router

.route("/all\_orphans\_of\_current\_orphanage/:id")

.get(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), checkOrphanage, orphanController.getAllOrphansOfOrphanage) // Get all orphans for orphanage of current orphanage admin

router

.route("/:orphanageid/:id")

.get(verifyToken, allowedTo(userRoles.ORPHANAGE\_ADMIN), checkOrphanage, orphanController.getOrphanByIdInOrphanage) // Get orphan by id in orphanage of current orphanage admin

module.exports = router;

**File: review.controller.js**

const Review = require('../models/review.model.js');

const Orphanage = require('../models/orphanage.model.js');

const appError = require('../utilities/appError.js');

const httpStatusText = require('../utilities/httpStatusText.js');

const asyncWrapper = require('../middlewares/asyncWrapper.js');

const sendEmail = require('../utilities/sendEmail.js');

// أضف تقييم جديد

const addReview = asyncWrapper(async (req, res, next) => {

const donorId = req.currentUser.id;

const { orphanageId, rating, comment } = req.body;

if (!orphanageId || !rating) {

return next(appError.create('Orphanage and rating are required', 400, httpStatusText.FAIL));

}

if (rating < 1 || rating > 5) {

return next(appError.create('Rating must be between 1 and 5', 400, httpStatusText.FAIL));

}

const orphanage = await Orphanage.findById(orphanageId);

if (!orphanage) {

return next(appError.create('Orphanage not found', 404, httpStatusText.FAIL));

}

// تحقق إذا كان التقييم موجود مسبقًا

const existingReview = await Review.findOne({ donor: donorId, orphanage: orphanageId });

if (existingReview) {

return next(appError.create('You have already reviewed this orphanage', 400, httpStatusText.FAIL));

}

const review = await Review.create({ donor: donorId, orphanage: orphanageId, rating, comment });

// إرسال رسالة شكر للمتبرع (مثال)

await sendEmail({

to: req.currentUser.email,

subject: 'شكراً لتقييمك',

text: `شكراً لتقييمك لمؤسسة ${orphanage.name} بدرجة ${rating} نجوم!`

});

res.status(201).json({

status: httpStatusText.SUCCESS,

message: 'Review added successfully',

data: { review }

});

});

// عرض تقييمات مؤسسة مع خيارات فرز

const getReviewsForOrphanage = asyncWrapper(async (req, res, next) => {

const { orphanageId } = req.params;

const { sortBy = 'createdAt', order = 'desc' } = req.query; // ترتيب: createdAt أو rating

if (!['createdAt', 'rating'].includes(sortBy)) {

return next(appError.create('Invalid sort field', 400, httpStatusText.FAIL));

}

if (!['asc', 'desc'].includes(order)) {

return next(appError.create('Invalid order', 400, httpStatusText.FAIL));

}

const orphanage = await Orphanage.findById(orphanageId);

if (!orphanage) {

return next(appError.create('Orphanage not found', 404, httpStatusText.FAIL));

}

const reviews = await Review.find({ orphanage: orphanageId })

.populate('donor', 'firstName lastName')

.sort({ [sortBy]: order === 'asc' ? 1 : -1 });

// حساب متوسط التقييم وعدد التقييمات

const agg = await Review.aggregate([

{ $match: { orphanage: orphanage.\_id } },

{

$group: {

\_id: '$orphanage',

averageRating: { $avg: '$rating' },

count: { $sum: 1 }

}

}

]);

const summary = agg[0] || { averageRating: 0, count: 0 };

res.json({

status: httpStatusText.SUCCESS,

data: { reviews, averageRating: summary.averageRating, totalReviews: summary.count }

});

});

// تحديث تقييم متبرع

const updateReview = asyncWrapper(async (req, res, next) => {

const donorId = req.currentUser.id;

const { reviewId } = req.params;

const { rating, comment } = req.body;

const review = await Review.findOne({ \_id: reviewId, donor: donorId });

if (!review) return next(appError.create('Review not found or unauthorized', 404, httpStatusText.FAIL));

if (rating !== undefined) {

if (rating < 1 || rating > 5) return next(appError.create('Rating must be between 1 and 5', 400, httpStatusText.FAIL));

review.rating = rating;

}

if (comment !== undefined) review.comment = comment;

await review.save();

res.json({

status: httpStatusText.SUCCESS,

message: 'Review updated successfully',

data: { review }

});

});

// حذف تقييم

const deleteReview = asyncWrapper(async (req, res, next) => {

const donorId = req.currentUser.id;

const { reviewId } = req.params;

const review = await Review.findOneAndDelete({ \_id: reviewId, donor: donorId });

if (!review) return next(appError.create('Review not found or unauthorized', 404, httpStatusText.FAIL));

res.json({

status: httpStatusText.SUCCESS,

message: 'Review deleted successfully'

});

});

const replyToReview = asyncWrapper(async (req, res, next) => {

const { reviewId } = req.params;

const { reply } = req.body;

const adminId = req.currentUser.id;

const orphanage = await Orphanage.findOne({ admin: adminId });

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

const review = await Review.findOne({ \_id: reviewId, orphanage: orphanage.\_id });

if (!review) {

return next(appError.create("Review not found or not related to your orphanage", 404, httpStatusText.FAIL));

}

review.reply = reply;

await review.save();

res.json({

status: httpStatusText.SUCCESS,

message: "Reply added successfully",

data: { review }

});

});

module.exports = {

addReview,

getReviewsForOrphanage,

updateReview,

deleteReview,

replyToReview

};

**File: review.model.js**

const mongoose = require('mongoose');

const ReviewSchema = new mongoose.Schema({

donor: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: 'Orphanage', required: true },

rating: { type: Number, required: true, min: 1, max: 5 },

comment: { type: String, default: '' },

reply: { type: String, default: null },

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

});

//اضمن انه المتبرع بقيم مرة وحدة للمؤسسة

ReviewSchema.index({ donor: 1, orphanage: 1 }, { unique: true });

module.exports = mongoose.model('Review', ReviewSchema);

**File: review.route.js**

const express = require('express');

const router = express.Router();

const verifyToken = require('../middlewares/verifyToken.js');

const allowedTo = require('../middlewares/allowedTo.js');

const userRoles = require('../utilities/userRoles.js');

const reviewController = require('../controllers/review.controller.js');

// إضافة تقييم جديد (متبرع فقط)

router.post(

'/',

verifyToken,

allowedTo(userRoles.DONOR),

reviewController.addReview

);

// جلب تقييمات مؤسسة معينة (عام)

router.get(

'/orphanage/:orphanageId',

reviewController.getReviewsForOrphanage

);

// تحديث تقييم (متبرع فقط)

router.patch(

'/:reviewId',

verifyToken,

allowedTo(userRoles.DONOR),

reviewController.updateReview

);

// حذف تقييم (متبرع فقط)

router.delete(

'/:reviewId',

verifyToken,

allowedTo(userRoles.DONOR),

reviewController.deleteReview

);

router.patch(

"/:reviewId/reply",

verifyToken,

allowedTo(userRoles.ORPHANAGE\_ADMIN),

reviewController.replyToReview

);

module.exports = router;

**File: sponsorship.model.js**

const mongoose = require("mongoose");

const sponsorshipStatus = require("../utilities/sponsorshipStatus.js");

const sponsorshipSchema = new mongoose.Schema(

{

sponsor: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

orphan: { type: mongoose.Schema.Types.ObjectId, ref: "Orphan", required: true },

amount: { type: Number, required: true, min: 1 },

currency: { type: String, enum: ["USD"] },

frequency: { type: String, enum: ["month", "year", "week", "day"], required: true },

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

endDate: { type: Date },

status: {

type: String,

enum: [sponsorshipStatus.PENDING, sponsorshipStatus.FAILD, sponsorshipStatus.ACTIVE, sponsorshipStatus.COMPLETED, sponsorshipStatus.CANCELED],

default: sponsorshipStatus.PENDING

},

subscriptionId: { type: String, required: true }, // ID from Stripe/PayPal subscription (for automatic payment)

latestInvoiceId: { type: String }, // Track latest invoice

},

{ timestamps: true }

);

sponsorshipSchema.index({ donor: 1, orphan: 1, startDate: 1 }); // Allows multiple sponsorships for an orphan by the same donor

module.exports = mongoose.model("Sponsorship", sponsorshipSchema);

**File: sponsorships.controller.js**

const mongoose = require("mongoose");

const Sponsorship = require("../models/sponsorship.model.js");

const User = require("../models/user.model.js");

const Orphan = require("../models/orphan.model.js");

const Orphanage = require("../models/orphanage.model.js");

const asyncWrapper = require("../middlewares/asyncWrapper.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const sponsorshipStatus = require("../utilities/sponsorshipStatus.js");

const stripe = require("stripe")(process.env.STRIPE\_SECRET\_KEY);

// Create Sponsorship with Subscription

const createSponsorship = asyncWrapper(async (req, res, next) => {

const { orphan, amount, currency, frequency, endDate } = req.body;

const sponsor = req.currentUser.id;

// Validate required fields

if (!sponsor || !orphan || !amount || !currency || !frequency || !endDate) {

return next(appError.create("Missing required fields", 400, httpStatusText.FAIL));

}

// Check if sponsor exists

const sponsorExists = await User.findById(sponsor);

if (!sponsorExists) {

return next(appError.create("Sponsor not found", 404, httpStatusText.FAIL));

}

// Check if orphan exists

const orphanExists = await Orphan.findById(orphan);

if (!orphanExists) {

return next(appError.create("Orphan not found", 404, httpStatusText.FAIL));

}

// Check orphanage existence

const orphanageExists = await Orphanage.findById(orphanExists.orphanage);

if (!orphanageExists) {

return next(appError.create("Orphanage not found or does not match the orphan", 400, httpStatusText.FAIL));

}

// Ensure the sponsor has a Stripe customer ID

if (!sponsorExists.stripeCustomerId) {

return res.status(400).json({ message: "Sponsor does not have a Stripe customer ID" });

}

// If the sponsor does not have a Stripe Customer ID, create one

// if (!sponsorExists.stripeCustomerId) {

// const customer = await stripe.customers.create({

// email: sponsorExists.email,

// name: sponsorExists.name,

// });

// sponsorExists.stripeCustomerId = customer.id;

// await sponsorExists.save();

// }

// Create a Stripe Subscription

// const price = await stripe.prices.create({

// unit\_amount: amount \* 100, // Convert to cents

// currency: currency.toLowerCase(),

// recurring: { interval: frequency.toLowerCase() },

// product: process.env.STRIPE\_PRODUCT\_ID,

// });

console.log('hello');

const subscription = await stripe.subscriptions.create({

customer: sponsorExists.stripeCustomerId,

items: [{ price: process.env.STRIPE\_PRODUCT\_PRICE\_ID }],

default\_payment\_method: sponsorExists.defaultPaymentMethodId, // Attach the saved card

payment\_behavior: "allow\_incomplete",

expand: ["latest\_invoice.payment\_intent"],

});

console.log('hi');

// Check if payment\_intent exists

const paymentIntent = subscription.latest\_invoice?.payment\_intent;

const nextActionUrl = paymentIntent?.next\_action?.use\_stripe\_sdk?.stripe\_js || null;

// Create Sponsorship entry

const sponsorship = new Sponsorship({

sponsor: sponsor,

orphan: orphan,

amount: amount,

currency: currency,

frequency: frequency,

endDate: endDate,

status: sponsorshipStatus.PENDING,

subscriptionId: subscription.id,

latestInvoiceId: subscription.latest\_invoice.id,

});

await sponsorship.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Sponsorship subscription created successfully",

data: { sponsorship, subscriptionUrl: nextActionUrl }

});

});

module.exports = { createSponsorship };

**File: sponsorships.route.js**

const express = require("express");

const router = express.Router();

const sponsorshipController = require("../controllers/sponsorships.controller.js");

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

const Sponsor = require("../models/user.model.js");

const stripe = require("stripe")(process.env.STRIPE\_SECRET\_KEY);

// Step 1: Create a Setup Intent & Generate Checkout URL

router.post("/setup-payment", verifyToken , allowedTo(userRoles.SPONSOR,userRoles.DONOR), async (req, res) => {

try {

console.log("setup");

const sponsorId = req.currentUser.id; // Get sponsor ID from request

// Fetch the sponsor from the database

const sponsor = await Sponsor.findById(sponsorId);

if (!sponsor) {

return res.status(404).json({ message: "Sponsor not found" });

}

// Ensure the sponsor has a Stripe customer ID

// if (!sponsor.stripeCustomerId) {

// return res.status(400).json({ message: "Sponsor does not have a Stripe customer ID" });

// }

//If the sponsor does not have a Stripe Customer ID, create one

if (!sponsor.stripeCustomerId) {

const customer = await stripe.customers.create({

email: sponsor.email,

name: sponsor.name,

});

sponsor.stripeCustomerId = customer.id;

await sponsor.save();

}

// Create a Setup Intent (for saving payment method)

const session = await stripe.checkout.sessions.create({

payment\_method\_types: ["card"],

mode: "setup",

customer: sponsor.stripeCustomerId,

success\_url: "https://yourapp.com/success",

cancel\_url: "https://yourapp.com/cancel",

});

res.json({ url: session.url }); // Send URL to frontend

} catch (error) {

console.error("Error creating Setup Intent:", error);

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

}

});

router.post("/attach-payment", verifyToken, allowedTo(userRoles.SPONSOR,userRoles.DONOR), async (req, res) => {

try {

const sponsorId = req.currentUser.id; // Get sponsor ID

// Fetch the sponsor from the database

const sponsor = await Sponsor.findById(sponsorId);

if (!sponsor) {

return res.status(404).json({ message: "Sponsor not found" });

}

// List payment methods for the customer

const paymentMethods = await stripe.paymentMethods.list({

customer: sponsor.stripeCustomerId,

type: "card",

});

if (paymentMethods.data.length === 0) {

return res.status(400).json({ message: "No payment method found" });

}

// Get the first payment method

const paymentMethodId = paymentMethods.data[0].id;

// Attach the payment method to the customer

await stripe.paymentMethods.attach(paymentMethodId, {

customer: sponsor.stripeCustomerId,

});

// Set as default payment method

await stripe.customers.update(sponsor.stripeCustomerId, {

invoice\_settings: { default\_payment\_method: paymentMethodId },

});

res.json({ message: "Payment method attached successfully" });

} catch (error) {

console.error("Error attaching payment method:", error);

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

}

});

router

.route("/:orphanageid")

.post(verifyToken, allowedTo(userRoles.SPONSOR), sponsorshipController.createSponsorship);

module.exports = router;

**File: supportProgram.controller.js**

// controllers/supportProgram.controller.js

const SupportProgram = require("../models/supportProgram.model.js");

const Orphanage = require("../models/orphanage.model");

const asyncWrapper = require("../middlewares/asyncWrapper");

const appError = require("../utilities/appError");

const httpStatusText = require("../utilities/httpStatusText");

const User = require("../models/user.model.js");

const userRoles = require('../utilities/userRoles');

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const createSupportProgram = asyncWrapper(async (req, res, next) => {

const { name, description, orphanage } = req.body;

const userId = req.currentUser.id;

if (!name) {

return next(appError.create("Program name is required", 400, httpStatusText.FAIL));

}

let orphanageRef = null;

if (orphanage) {

const exists = await Orphanage.findById(orphanage);

if (!exists) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

orphanageRef = orphanage;

}

const program = new SupportProgram({

name,

description,

orphanage: orphanageRef,

createdBy: userId,

});

await program.save();

if (orphanageRef) {

await Orphanage.findByIdAndUpdate(

orphanageRef,

{ $push: { supportPrograms: program.\_id } },

{ new: true }

);

}

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Support Program created successfully",

data: { program },

});

});

// const getAllSupportPrograms = asyncWrapper(async (req, res, next) => {

// console.log("wafa");

// const programs = await SupportProgram.find()

// .populate("orphanage", "name location")

// .populate("createdBy", "firstName lastName email");

// res.status(200).json({

// status: httpStatusText.SUCCESS,

// data: { programs },

// });

// });

const getAllSupportPrograms = asyncWrapper(async (req, res, next) => {

const userId = req.currentUser.id;

console.log("User ID:", userId);

const user = await User.findById(userId);

if (!user) {

return next(appError.create("User not found", 404, httpStatusText.FAIL));

}

const userRole = user.role;

if (userRole === "ORPHANAGE ADMIN") {

const orphanage = await Orphanage.findOne({ admin: userId });

if (!orphanage) {

return next(appError.create("Orphanage not found for this admin", 404, httpStatusText.FAIL));

}

const programs = await SupportProgram.find({ orphanage: orphanage.\_id })

.populate("orphanage", "name location")

.populate("createdBy", "firstName lastName email");

return res.status(200).json({

status: httpStatusText.SUCCESS,

data: { programs },

});

} else {

// إذا كان المستخدم "Admin" فقط، يمكنه رؤية كل البرامج

const programs = await SupportProgram.find()

.populate("orphanage", "name location")

.populate("createdBy", "firstName lastName email");

return res.status(200).json({

status: httpStatusText.SUCCESS,

data: { programs },

});

}

});

const getSupportProgramById = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const userId = req.currentUser.id;

const userRoles = req.currentUser.role;

// البحث عن البرنامج

const program = await SupportProgram.findById(id)

.populate("orphanage", "name location")

.populate("createdBy", "firstName lastName email");

if (!program) {

return next(appError.create("Support Program not found", 404, httpStatusText.FAIL));

}

// إذا كان المستخدم "Orphanage Admin"، تأكد أن البرنامج مرتبط بمؤسسته

if (userRoles === "ORPHANAGE\_ADMIN") {

const orphanage = await Orphanage.findOne({ admin: userId });

if (!orphanage || orphanage.\_id.toString() !== program.orphanage.toString()) {

return next(appError.create("Unauthorized - You are not the admin of this orphanage", 403, httpStatusText.FAIL));

}

}

res.status(200).json({

status: httpStatusText.SUCCESS,

data: { program },

});

});

const getSupportProgramByOrphanage = asyncWrapper(async (req, res, next) => {

const { orphanageId } = req.params; // جلب الـ orphanageId من المعاملات

const userId = req.currentUser.id; // استرجاع الـ userId من الـ currentUser

// استرجاع بيانات المستخدم

const user = await User.findById(userId);

if (!user) {

return next(appError.create("User not found", 404, httpStatusText.FAIL));

}

const userRole = user.role; // استرجاع الدور من بيانات المستخدم

// إذا كان المستخدم هو "Orphanage Admin"

if (userRole === "ORPHANAGE ADMIN") {

const orphanage = await Orphanage.findById(orphanageId);

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

// تحقق مما إذا كان الـ orphanage الخاص بالمستخدم يطابق الـ orphanage المطلوب

if (!orphanage.admin.equals(userId)) {

return next(appError.create("Unauthorized - You are not the admin of this orphanage", 403, httpStatusText.FAIL));

}

// جلب جميع البرامج المرتبطة بالمؤسسة

const programs = await SupportProgram.find({ orphanage: orphanageId })

.populate("orphanage", "name location")

.populate("createdBy", "firstName lastName email");

return res.status(200).json({

status: httpStatusText.SUCCESS,

data: { programs },

});

}

// إذا كان المستخدم "SUPPORT\_PROGRAM\_ADMIN"

else if (userRole === "SUPPORT\_PROGRAM\_ADMIN") {

// جلب البرامج التي تم إنشاؤها بواسطة الـ SUPPORT\_PROGRAM\_ADMIN لهذا الـ orphanage

const programs = await SupportProgram.find({ orphanage: orphanageId, createdBy: userId })

.populate("orphanage", "name location")

.populate("createdBy", "firstName lastName email");

return res.status(200).json({

status: httpStatusText.SUCCESS,

data: { programs },

});

}

// إذا كان المستخدم "Admin" فقط، يمكنه رؤية جميع البرامج من جميع المؤسسات

else {

const programs = await SupportProgram.find({ orphanage: orphanageId })

.populate("orphanage", "name location")

.populate("createdBy", "firstName lastName email");

return res.status(200).json({

status: httpStatusText.SUCCESS,

data: { programs },

});

}

});

// Delete support program (by owner only)

const deleteSupportProgram = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const program = await SupportProgram.findById(id);

if (!program) {

return next(appError.create("Support Program not found", 404, httpStatusText.FAIL));

}

if (!program.createdBy.equals(req.currentUser.id)) {

return next(appError.create("Unauthorized - you didn't create this program", 403, httpStatusText.FAIL));

}

await SupportProgram.findByIdAndDelete(id);

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Support Program deleted successfully"

});

});

module.exports = {

createSupportProgram,

getAllSupportPrograms,

getSupportProgramById,

deleteSupportProgram,

getSupportProgramByOrphanage

};

**File: supportProgram.model.js**

// models/supportProgram.model.js

const mongoose = require("mongoose");

const supportProgramSchema = new mongoose.Schema({

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

description: { type: String },

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage", required: false }, // optional

createdBy: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

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

});

module.exports = mongoose.model("SupportProgram", supportProgramSchema);

**File: supportProgram.route.js**

const express = require("express");

const router = express.Router();

const supportProgramController = require("../controllers/supportProgram.controller");

const verifyToken = require("../middlewares/verifyToken");

const allowedTo = require("../middlewares/allowedTo");

const userRoles = require("../utilities/userRoles");

// إنشاء Support Program

router.post(

"/create",

verifyToken,

allowedTo(userRoles.SUPPORT\_PROGRAM\_ADMIN),

supportProgramController.createSupportProgram

);

// الحصول على جميع برامج الدعم

router.get("/", verifyToken, allowedTo(userRoles.SUPPORT\_PROGRAM\_ADMIN,userRoles.ADMIN,userRoles.ORPHANAGE\_ADMIN), supportProgramController.getAllSupportPrograms);

// الحصول على برنامج دعم بواسطة ID

router.get("/:id", verifyToken ,allowedTo(userRoles.SUPPORT\_PROGRAM\_ADMIN, userRoles.ADMIN,userRoles.ORPHANAGE\_ADMIN),supportProgramController.getSupportProgramById);

// حذف برنامج الدعم

router.delete(

"/:id",

verifyToken,

allowedTo(userRoles.SUPPORT\_PROGRAM\_ADMIN),

supportProgramController.deleteSupportProgram

);

router.get(

"/orphanage/:orphanageId", // استخدام :orphanageId لتمرير معرف المؤسسة

verifyToken,

allowedTo(userRoles.SUPPORT\_PROGRAM\_ADMIN,userRoles.ADMIN, userRoles.ORPHANAGE\_ADMIN), // السماح للـ Admin و Orphanage Admin

supportProgramController.getSupportProgramByOrphanage

);

module.exports = router;

**File: systemSettings.controller.js**

const SystemSettings = require("../models/systemSettings.model.js");

const appError = require('../utilities/appError.js');

const httpStatusText = require('../utilities/httpStatusText.js');

const asyncWrapper = require('../middlewares/asyncWrapper.js');

const createDefaultSystemSettings = asyncWrapper(async (req, res, next) => {

const existing = await SystemSettings.findOne();

if (existing) {

return res.status(200).json({

status: httpStatusText.SUCCESS,

message: "SystemSettings already exists.",

data: { settings: existing }

});

}

const newSettings = await SystemSettings.create({ transactionFeePercent: 5 });

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Default SystemSettings created.",

data: { settings: newSettings }

});

});

const updateOperationalFee = asyncWrapper(async (req, res, next) => {

const { operationalFeePercentage } = req.body;

if (operationalFeePercentage < 0 || operationalFeePercentage > 0.3) {

return next(appError.create("Fee must be between 0 and 0.3", 400, httpStatusText.FAIL));

}

let settings = await SystemSettings.findOne();

if (!settings) {

settings = await SystemSettings.create({ operationalFeePercentage });

} else {

settings.operationalFeePercentage = operationalFeePercentage;

await settings.save();

}

res.json({

status: httpStatusText.SUCCESS,

message: "Operational fee updated successfully",

data: { operationalFeePercentage: settings.operationalFeePercentage }

});

});

const getSettings = asyncWrapper(async (req, res) => {

const settings = await SystemSettings.findOne();

res.json({

status: httpStatusText.SUCCESS,

data: settings || { operationalFeePercentage: 0.05 }

});

});

module.exports = {

createDefaultSystemSettings,

updateOperationalFee,

getSettings };

**File: systemSettings.model.js**

const mongoose = require("mongoose");

const systemSettingsSchema = new mongoose.Schema({

transactionFeePercent: {

type: Number,

required: true,

default: 5

}

});

module.exports = mongoose.model("SystemSettings", systemSettingsSchema);

**File: systemSettings.route.js**

// routes/systemSettings.route.js

const express = require("express");

const router = express.Router();

const verifyToken = require("../middlewares/verifyToken.js");

const allowedTo = require("../middlewares/allowedTo.js");

const userRoles = require("../utilities/userRoles.js");

const controller = require("../controllers/systemSettings.controller.js");

router.post("/init", controller.createDefaultSystemSettings);

router.get("/", verifyToken, allowedTo(userRoles.ADMIN), controller.getSettings);

router.patch("/", verifyToken, allowedTo(userRoles.ADMIN), controller.updateOperationalFee);

module.exports = router;

**File: uploadOrphanPhotos.js**

const multer = require("multer");

const path = require("path");

const storage = multer.diskStorage({

destination: (req, file, cb) => {

cb(null, "uploads/orphans/");

},

filename: (req, file, cb) => {

const ext = path.extname(file.originalname);

const name = `orphan-${Date.now()}-${Math.round(Math.random() \* 1e9)}${ext}`;

cb(null, name);

}

});

const fileFilter = (req, file, cb) => {

const allowedTypes = /jpeg|jpg|png/;

const isValid = allowedTypes.test(file.mimetype);

if (isValid) cb(null, true);

else cb(new Error("Only images are allowed"));

};

const upload = multer({

storage,

fileFilter,

limits: { fileSize: 5 \* 1024 \* 1024 } // 5MB max per file

});

module.exports = upload;

**File: uploadUserAvatar.js**

const multer = require("multer");

const path = require("path");

const storage = multer.diskStorage({

destination: (req, file, cb) => {

cb(null, "uploads/users/");

},

filename: (req, file, cb) => {

const ext = path.extname(file.originalname);

const name = `user-${Date.now()}-${Math.round(Math.random() \* 1e9)}${ext}`;

cb(null, name);

}

});

const fileFilter = (req, file, cb) => {

const allowedTypes = /jpeg|jpg|png/;

const isValid = allowedTypes.test(file.mimetype);

if (isValid) cb(null, true);

else cb(new Error("Only images are allowed"));

};

const upload = multer({

storage,

fileFilter,

limits: { fileSize: 5 \* 1024 \* 1024 } // 5MB max per file

});

module.exports = upload;

**File: user.model.js**

const mongoose = require('mongoose');

const validator = require('validator');

const userRoles = require('../utilities/userRoles');

const UserSchema = new mongoose.Schema({

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

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

email: { type: String, required: true, unique: true, validate: [validator.isEmail /\*we dont want to call the func (just pass it) thats why we did not put ()\*/, 'Field must be a valid email address'] },

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

phone: { type: String },

address: { type: String },

role: { type: String, enum: [userRoles.DONOR, userRoles.VOLUNTEER, userRoles.SPONSOR, userRoles.ORPHANAGE\_ADMIN, userRoles.ADMIN, userRoles.DRIVER, userRoles.SUPPORT\_PROGRAM\_ADMIN , userRoles.TEMPORARY], default: userRoles.DONOR, required: true },

driverStatus: { type: String, enum: ["BUSY", "AVAILABLE"], default: "AVAILABLE" },

driverCurrentLocation: {

type: {

type: String,

enum: ["Point"],

default: "Point"

},

coordinates: {

type: [Number],

default: [0, 0]

}

},

avatar: { type: String, default: 'uploads/profile.png' },

token: { type: String },

stripeCustomerId: { type: String }, // for sponsors

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

});

module.exports = mongoose.model("User", UserSchema);

**File: users.controller.js**

const asyncWrapper = require('../middlewares/asyncWrapper.js');

const User = require('../models/user.model.js');

const httpStatusText = require('../utilities/httpStatusText.js');

const appError = require('../utilities/appError.js');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const generateJWT = require('../utilities/generateJWT.js');

const userRoles = require('../utilities/userRoles.js');

const getAllUsers = asyncWrapper(

async (req,res) => {

console.log(req.headers);

const query = req.query;

/\* Pagination \*/

const limit = query.limit || 4;

const page = query.page || 1;

const skip = (page - 1) \* limit;

const users = await User.find({}, {\_\_v: false, 'password': false}).limit(limit).skip(skip);

/\* Pagination \*/

return res.json({ status: httpStatusText.SUCCESS, data: {users} });

}

);

const register = asyncWrapper(

async(req,res,next) => {

const {firstName, lastName, email, password, phone, address, role} = req.body;

let resultRole = role;

// Check if all required fields are provided

if (!firstName || !lastName || !email || !password || !phone || !address || !role) {

const error = appError.create('All fields are required', 400, httpStatusText.FAIL);

return next(error);

}

// Check if the email is valid

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!emailRegex.test(email)) {

const error = appError.create('Invalid email format', 400, httpStatusText.FAIL);

return next(error);

}

// Check if the password is strong

const passwordRegex = /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/;

if (!passwordRegex.test(password)) {

const error = appError.create('Password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, one number, and one special character', 400, httpStatusText.FAIL);

return next(error);

}

// Check if the phone number is valid

const phoneRegex = /^\d{10}$/;

if (!phoneRegex.test(phone)) {

const error = appError.create('Invalid phone number format', 400, httpStatusText.FAIL);

return next(error);

}

// Cheak if the role is ADMIN => if yes, then mark role as TEMPORARY

if (role === userRoles.ADMIN) {

resultRole = userRoles.TEMPORARY;

}

// Check if the role is valid

if (![userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN, userRoles.DRIVER, userRoles.ORPHANAGE\_ADMIN, userRoles.SPONSOR, userRoles.VOLUNTEER, userRoles.TEMPORARY].includes(resultRole)) {

const error = appError.create('Invalid role', 400, httpStatusText.FAIL);

return next(error);

}

// Check if file exists

let avatarFilename = null;

if (req.file) {

avatarFilename = req.file.filename;

}

// password hashing:

const hashedPassword = await bcrypt.hash(password, 10); // hash(password, salt /\*adding random string - to protect against rainbow table anmd brute-force attacks\*/)

const isDriver = (resultRole === userRoles.DRIVER);

const newUser = new User({

firstName,

lastName,

email,

password: hashedPassword,

phone,

address,

role: resultRole,

driverStatus: isDriver ? 'AVAILABLE' : undefined,

driverCurrentLocation: isDriver ? { type: 'Point', coordinates: [0, 0] } : undefined,

avatar: avatarFilename

});

// generate jwt token

const token = await generateJWT({id: newUser.\_id, email: newUser.email, role: newUser.role});

newUser.token = token;

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

if(oldUser) {

const error = appError.create('user already exists', 400, httpStatusText.FAIL);

return next(error);

}

await newUser.save();

return res.status(201).json({ status: httpStatusText.SUCCESS, data: {user: newUser} });

}

);

const login = asyncWrapper(

async (req, res, next) => {

const {email, password} = req.body;

if(!email || !password){

const error = appError.create('email and password are required', 400, httpStatusText.FAIL);

return next(error);

}

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

if(!user){

const error = appError.create('user not found', 400, httpStatusText.FAIL);

return next(error);

}

const matchedPassword = bcrypt.compare(password, user.password /\*hashed pass from DB\*/);

if(user && matchedPassword){

// logged in successfully

const token = await generateJWT({id: user.\_id, email: user.email, role: user.role});

return res.json({ status: httpStatusText.SUCCESS, data: {id: user.\_id, token}});

}

else{

const error = appError.create('invalid email or password', 404, httpStatusText.ERROR);

return next(error);

}

}

);

// ADMIN accepts/denies user registration with role ADMIN

const acceptAdministration = asyncWrapper(async (req, res, next) => {

const {id} = req.params;

const {status} = req.body;

const user = await User.findById(id);

if (!user) {

return next(appError.create('User not found', 404, httpStatusText.FAIL));

}

if (status === 'ACCEPT') {

user.role = userRoles.ADMIN;

}

else if (status === 'DENY') {

user.role = userRoles.DONOR;

}

else {

return next(appError.create('Invalid status', 400, httpStatusText.FAIL));

}

await user.save();

res.status(200).json({ status: httpStatusText.SUCCESS, message: `User ${status === 'ACCEPT' ? 'accepted' : 'denied'} successfully.` });

});

// Get all users with role TEMPORARY

const getAllTemporaryUsers = asyncWrapper(async (req, res, next) => {

const users = await User.find({ role: userRoles.TEMPORARY }, { \_\_v: false, password: false });

if (!users || users.length === 0) {

return next(appError.create('No temporary users found', 404, httpStatusText.FAIL));

}

res.status(200).json({ status: httpStatusText.SUCCESS, data: { users } });

});

// Update user information

const updateInfo = asyncWrapper(async (req, res, next) => {

const userId = req.currentUser.id;

const {firstName, lastName, email, password, phone, address, role} = req.body;

let resultRole = role;

// Check if all required fields are provided

if (!firstName || !lastName || !email || !password || !phone || !address || !role) {

const error = appError.create('All fields are required', 400, httpStatusText.FAIL);

return next(error);

}

// Check if the email is valid

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!emailRegex.test(email)) {

const error = appError.create('Invalid email format', 400, httpStatusText.FAIL);

return next(error);

}

// Check if the password is strong

const passwordRegex = /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/;

if (!passwordRegex.test(password)) {

const error = appError.create('Password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, one number, and one special character', 400, httpStatusText.FAIL);

return next(error);

}

// Check if the phone number is valid

const phoneRegex = /^\d{10}$/;

if (!phoneRegex.test(phone)) {

const error = appError.create('Invalid phone number format', 400, httpStatusText.FAIL);

return next(error);

}

// Cheak if the role is ADMIN => if yes, then mark role as TEMPORARY

if (role === userRoles.ADMIN) {

resultRole = userRoles.TEMPORARY;

}

// Check if the role is valid

if (![userRoles.DONOR, userRoles.SUPPORT\_PROGRAM\_ADMIN, userRoles.DRIVER, userRoles.ORPHANAGE\_ADMIN, userRoles.SPONSOR, userRoles.VOLUNTEER, userRoles.TEMPORARY].includes(resultRole)) {

const error = appError.create('Invalid role', 400, httpStatusText.FAIL);

return next(error);

}

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

if (existingUser && existingUser.\_id.toString() !== userId) {

return next(appError.create('Email already used by another user', 400, httpStatusText.FAIL));

}

// Check if file exists

let avatarFilename = null;

if (req.file) {

avatarFilename = req.file.filename;

}

// password hashing:

const hashedPassword = await bcrypt.hash(password, 10); // hash(password, salt /\*adding random string - to protect against rainbow table anmd brute-force attacks\*/)

const isDriver = (resultRole === userRoles.DRIVER);

// generate jwt token

const token = await generateJWT({id: userId, email: email, role: role});

const user = await User.findByIdAndUpdate(

userId,

{

firstName,

lastName,

email,

password: hashedPassword,

phone,

address,

role: resultRole,

token,

driverStatus: isDriver ? 'AVAILABLE' : undefined,

driverCurrentLocation: isDriver ? { type: 'Point', coordinates: [0, 0] } : undefined,

avatar: avatarFilename

},

{ new: true, runValidators: true }

);

if (!user) return next(appError.create("User not found", 404, httpStatusText.FAIL));

res.status(200).json({ status: httpStatusText.SUCCESS, data: { user } });

});

// Delete user account

const deleteMyAccount = asyncWrapper(async (req, res, next) => {

const userId = req.currentUser.id;

const user = await User.findByIdAndDelete(userId);

if (!user) return next(appError.create("User not found", 404, httpStatusText.FAIL));

res.status(200).json({ status: httpStatusText.SUCCESS, message: "Your account has been deleted." });

});

// Delete user by ID (admin only)

const deleteUserById = asyncWrapper(async (req, res, next) => {

const userId = req.params.id;

const user = await User.findByIdAndDelete(userId);

if (!user) return next(appError.create("User not found", 404, httpStatusText.FAIL));

res.status(200).json({ status: httpStatusText.SUCCESS, message: "User deleted successfully." });

});

// get user by ID

const getUserById = asyncWrapper(async (req, res, next) =>

{

const userId = req.params.id;

const user = await User.findById(userId, { \_\_v: false, password: false, avatar: false });

if (!user) return next(appError.create("User not found", 404, httpStatusText.FAIL));

res.status(200).json({ status: httpStatusText.SUCCESS, data: { user } });

}

);

module.exports = {

getAllUsers,

register,

login,

updateInfo,

deleteMyAccount,

deleteUserById,

acceptAdministration,

getAllTemporaryUsers,

getUserById

}

**File: users.route.js**

const express = require('express');

const router = express.Router();

const usersController = require('../controllers/users.controller.js');

const verifyToken = require('../middlewares/verifyToken.js');

const appError = require('../utilities/appError.js');

const httpStatusText = require('../utilities/httpStatusText.js');

const upload = require('../middlewares/uploadUserAvatar.js');

const allowedTo = require('../middlewares/allowedTo.js');

const userRoles = require('../utilities/userRoles.js');

router.route('/')

.get(verifyToken, usersController.getAllUsers)

.put(

verifyToken,

upload.single("avatar"),

usersController.updateInfo

)

.delete(verifyToken, usersController.deleteMyAccount);

router.route('/register')

.post(upload.single('avatar'), usersController.register);

router.route('/login')

.post(usersController.login);

//Get All Users With Role TEMPORARY

router.route('/temporary')

.get(verifyToken, allowedTo(userRoles.ADMIN), usersController.getAllTemporaryUsers);

router.route('/:id')

.delete(verifyToken, usersController.deleteUserById)

.get(verifyToken, usersController.getUserById);

//Accept Adminitration Request

router.route('/AdministrationRequest/:id')

.put(verifyToken, allowedTo(userRoles.ADMIN), usersController.acceptAdministration);

//Forget Password

// router.route('/forgetPassword')

// .post(usersController.forgetPassword);

module.exports = router;

**File: validationSchema.js**

const {body} = require('express-validator');

const validationSchema = () => {

return [

body('name')

.notEmpty()

.withMessage("Name is at least 1 character")

.isLength({min: 1})

.withMessage("Name is required")

.isString()

.withMessage("Name is string"),

body('price')

.notEmpty()

.withMessage("Price is required")

.isNumeric()

.withMessage("Price is numeric") /\*chaining\*/

];

};

module.exports = {

validationSchema

};

**File: verifyToken.js**

const jwt = require('jsonwebtoken');

const httpStatusText = require('../utilities/httpStatusText.js');

const appError = require('../utilities/appError.js');

const verifyToken = (req, res, next) => {

const authHeader = req.headers['Authorization'] || req.headers['authorization'];

if(!authHeader) {

return res.status(401).json({ message: 'token is rquired' });

}

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

try{

const currentUser = jwt.verify(token, process.env.JWT\_SECRET\_KEY); // decode

req.currentUser = currentUser;

next();

}

catch(err){

const error = appError.create("Invalid token", 401, httpStatusText.ERROR);// unauthorized

return next(error);

}

}

module.exports = verifyToken;

**File: VolunteerApplication.model.js**

const mongoose = require("mongoose");

const VolunteerApplicationSchema = new mongoose.Schema({

volunteer: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true },

volunteerRequest: { type: mongoose.Schema.Types.ObjectId, ref: "VolunteerRequest", required: true },

serviceType: {

type: String,

enum: ["Teaching", "Mentoring", "Healthcare", "Other"],

required: true

},

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage", required: true },

message: { type: String },

status: { type: String, enum: ["Pending", "Accepted", "Rejected"], default: "Pending" },

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

});

module.exports = mongoose.model("VolunteerApplication", VolunteerApplicationSchema);

**File: VolunteerApplications.controller.js**

const VolunteerApplication = require("../models/VolunteerApplication.model.js");

const VolunteerRequest = require("../models/VolunteerRequest.model.js");

const Orphanage = require("../models/orphanage.model.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const asyncWrapper = require("../middlewares/asyncWrapper.js");

// ✅ Volunteer applies to a request

const applyToVolunteerRequest = asyncWrapper(async (req, res, next) => {

const { requestId } = req.params;

const volunteerId = req.currentUser.id;

const { serviceType, message } = req.body;

if (!["Teaching", "Mentoring", "Healthcare", "Other"].includes(serviceType)) {

return next(appError.create("Invalid service type", 400, httpStatusText.FAIL));

}

const request = await VolunteerRequest.findById(requestId).populate("orphanage");

if (!request || request.status === "Closed" ) {

return next(appError.create("Volunteer request is not available", 400, httpStatusText.FAIL));

}

const existing = await VolunteerApplication.findOne({

volunteer: volunteerId,

volunteerRequest: requestId

});

if (existing) {

return next(appError.create("You already applied to this request", 400, httpStatusText.FAIL));

}

const application = await VolunteerApplication.create({

volunteer: volunteerId,

volunteerRequest: request.\_id,

orphanage: request.orphanage.\_id,

serviceType,

message

});

request.applicationCount += 1;

if (request.applicationCount >= request.maxVolunteers) {

request.status = "Closed";

}

await request.save();

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Application submitted successfully.",

data: { application }

});

});

// ✅ Volunteer gets their applications

const getMyApplications = asyncWrapper(async (req, res, next) => {

const volunteerId = req.currentUser.id;

const applications = await VolunteerApplication.find({ volunteer: volunteerId })

.populate("volunteerRequest", "description requiredServiceType status")

.populate("orphanage", "name");

// تعديل كل تطبيق حسب حالة الطلب المرتبط

const modifiedApplications = applications.map(app => {

const appObject = app.toObject();

if (!app.volunteerRequest) {

// تم حذف الطلب المرتبط

appObject.volunteerRequest = {

status: "Deleted",

description: "This request was deleted by the orphanage admin.",

requiredServiceType: "N/A"

};

}

return appObject;

});

res.json({

status: httpStatusText.SUCCESS,

data: { applications: modifiedApplications }

});

});

// ✅ Orphanage admin views applications

const getApplicationsForOrphanage = asyncWrapper(async (req, res, next) => {

const adminId = req.currentUser.id;

const orphanage = await Orphanage.findOne({ admin: adminId });

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

const filter = { orphanage: orphanage.\_id };

// فلترة حسب نوع الخدمة إذا تم تحديدها

if (req.query.serviceType) {

filter.serviceType = req.query.serviceType;

}

const applications = await VolunteerApplication.find({ orphanage: orphanage.\_id })

.populate("volunteer", "firstName lastName email phone")

.populate("volunteerRequest", "description requiredServiceType");

res.json({

status: httpStatusText.SUCCESS,

data: { applications }

});

});

// ✅ Approve application

const approveApplication = asyncWrapper(async (req, res, next) => {

const { applicationId } = req.params;

const application = await VolunteerApplication.findById(applicationId).populate("volunteerRequest");

if (!application) {

return next(appError.create("Application not found", 404, httpStatusText.FAIL));

}

if (application.status !== "Pending") {

return next(appError.create("Application already processed", 400, httpStatusText.FAIL));

}

application.status = "Accepted";

await application.save();

// ✅ Increment acceptedCount

const request = await VolunteerRequest.findById(application.volunteerRequest.\_id);

request.acceptedCount = (request.acceptedCount || 0) + 1;

if (request.acceptedCount >= request.maxVolunteers) {

request.status = "Closed";

}

await request.save();

res.json({

status: httpStatusText.SUCCESS,

message: "Application approved.",

data: { application }

});

});

// ✅ Reject application

const rejectApplication = asyncWrapper(async (req, res, next) => {

const { applicationId } = req.params;

const application = await VolunteerApplication.findById(applicationId).populate("volunteerRequest");

if (!application) {

return next(appError.create("Application not found", 404, httpStatusText.FAIL));

}

if (application.status !== "Pending" && application.status !== "Accepted") {

return next(appError.create("Application already processed", 400, httpStatusText.FAIL));

}

// Decrement acceptedCount if previously accepted

if (application.status === "Accepted") {

const request = await VolunteerRequest.findById(application.volunteerRequest.\_id);

request.acceptedCount = Math.max(0, (request.acceptedCount || 0) - 1);

if (request.status === "Closed" && request.acceptedCount < request.maxVolunteers) {

request.status = "Open";

}

await request.save();

}

application.status = "Rejected";

await application.save();

res.json({

status: httpStatusText.SUCCESS,

message: "Application rejected.",

data: { application }

});

});

// ✅ Get all approved orphanages (for volunteers)

const getAllApprovedOrphanages = asyncWrapper(async (req, res) => {

const orphanages = await Orphanage.find({ status: "APPROVED", verified: true }, { \_\_v: 0 });

res.json({

status: httpStatusText.SUCCESS,

data: { orphanages }

});

});

// ✅ Get all open volunteer requests

const getAllOpenRequests = asyncWrapper(async (req, res) => {

const requests = await VolunteerRequest.find({ status: "Open" })

.populate("orphanage", "name location")

.select("-\_\_v");

res.json({

status: httpStatusText.SUCCESS,

data: { requests }

});

});

// ✅ Get available service types

const getServiceTypes = asyncWrapper(async (req, res) => {

const types = ["Teaching", "Mentoring", "Healthcare", "Other"];

res.json({

status: httpStatusText.SUCCESS,

data: { serviceTypes: types }

});

});

const getVolunteerRequestsByOrphanageId = asyncWrapper(async (req, res, next) => {

const { orphanageId } = req.params;

const { limit = 4, page = 1 } = req.query;

const skip = (page - 1) \* limit;

const filter = {

orphanage: orphanageId,

status: "Open"

};

if (req.query.serviceType) {

filter.requiredServiceType = req.query.serviceType;

}

const requests = await VolunteerRequest.find(filter)

.skip(skip)

.limit(Number(limit))

.populate("orphanage", "name location");

res.json({

status: httpStatusText.SUCCESS,

data: { requests }

});

});

// const deleteVolunteerApplication = asyncWrapper(async (req, res, next) => {

// const { applicationId } = req.params;

// const currentUserId = req.currentUser.id;

// const application = await VolunteerApplication.findById(applicationId).populate("volunteerRequest");

// if (!application) {

// return next(appError.create("Application not found", 404, httpStatusText.FAIL));

// }

// // السماح بالحذف فقط إذا كان المتطوع هو صاحب الطلب

// if (application.volunteer.toString() !== currentUserId) {

// return next(appError.create("Not authorized to delete this application", 403, httpStatusText.FAIL));

// }

// const request = application.volunteerRequest;

// // إذا كان التطبيق مقبول، ننقص acceptedCount

// if (application.status === "Accepted") {

// request.acceptedCount = Math.max((request.acceptedCount || 1) - 1, 0);

// if (request.status === "Closed" && request.acceptedCount < request.maxVolunteers) {

// request.status = "Open";

// }

// await request.save();

// }

// await application.deleteOne();

// res.json({

// status: httpStatusText.SUCCESS,

// message: "Application deleted successfully."

// });

// });

const deleteVolunteerApplication = asyncWrapper(async (req, res, next) => {

const { applicationId } = req.params;

const currentUserId = req.currentUser.id;

const application = await VolunteerApplication.findById(applicationId).populate("volunteerRequest");

if (!application) {

return next(appError.create("Application not found", 404, httpStatusText.FAIL));

}

if (!application.volunteer.equals(currentUserId)) {

return next(appError.create("Not authorized to delete this application", 403, httpStatusText.FAIL));

}

const request = application.volunteerRequest;

if (!request) {

return next(appError.create("Related volunteer request not found", 404, httpStatusText.FAIL));

}

if (application.status === "Accepted") {

request.acceptedCount = Math.max((request.acceptedCount || 0) - 1, 0);

if (request.status === "Closed" && request.acceptedCount < request.maxVolunteers) {

request.status = "Open";

}

await request.save();

}

await application.deleteOne();

res.json({

status: httpStatusText.SUCCESS,

message: "Application deleted successfully."

});

});

module.exports = {

applyToVolunteerRequest,

getMyApplications,

getApplicationsForOrphanage,

approveApplication,

rejectApplication,

getAllApprovedOrphanages,

getAllOpenRequests,

getServiceTypes,

getVolunteerRequestsByOrphanageId,

deleteVolunteerApplication

};

**File: volunteerApplications.route.js**

const express = require("express");

const router = express.Router();

const verifyToken = require("../middlewares/verifyToken");

const allowedTo = require("../middlewares/allowedTo");

const userRoles = require("../utilities/userRoles");

const controller = require("../controllers/VolunteerApplications.controller.js");

// 👤 Only VOLUNTEER

// تقديم طلب تطوع لطلب معين

router.post(

"/apply/:requestId",

verifyToken,

allowedTo(userRoles.VOLUNTEER),

controller.applyToVolunteerRequest

);

// عرض كل طلبات المتطوع الحالي

router.get(

"/my-applications",

verifyToken,

allowedTo(userRoles.VOLUNTEER),

controller.getMyApplications

);

router.get(

"/orphanages",

verifyToken,

allowedTo(userRoles.VOLUNTEER),

controller.getAllApprovedOrphanages

);

// عرض كل الطلبات المفتوحة

router.get(

"/requests",

verifyToken,

allowedTo(userRoles.VOLUNTEER),

controller.getAllOpenRequests

);

// عرض أنواع الخدمات التطوعية

router.get(

"/service-types",

verifyToken,

allowedTo(userRoles.VOLUNTEER),

controller.getServiceTypes

);

router.get(

"/volunteer-requests/orphanage/:orphanageId",

verifyToken,

allowedTo(userRoles.VOLUNTEER),

controller.getVolunteerRequestsByOrphanageId

);

router.delete(

"/:applicationId",

verifyToken,

allowedTo(userRoles.VOLUNTEER),

controller.deleteVolunteerApplication

);

module.exports = router;

**File: volunteerRequest.model.js**

const mongoose = require('mongoose');

const VolunteerRequestSchema = new mongoose.Schema({

orphanage: { type: mongoose.Schema.Types.ObjectId, ref: "Orphanage", required: true },

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

requiredSkills: { type: String },

requiredServiceType: {

type: String,

enum: ["Teaching", "Mentoring", "Healthcare", "Other"],

required: true

},

status: { type: String, enum: ["Open", "Closed"], default: "Open" },

applicationCount: { type: Number, default: 0 },

maxVolunteers: { type: Number, default: 5 },

acceptedCount: { type: Number, default: 0 },

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

});

// module.exports = mongoose.model("VolunteerRequest", VolunteerRequestSchema);

module.exports = mongoose.models.VolunteerRequest || mongoose.model("VolunteerRequest", VolunteerRequestSchema);

**File: VolunteerRequests.controller.js**

const VolunteerRequest = require("../models/volunteerRequest.model.js");

const Orphanage = require("../models/orphanage.model.js");

const appError = require("../utilities/appError.js");

const httpStatusText = require("../utilities/httpStatusText.js");

const asyncWrapper = require("../middlewares/asyncWrapper.js");

const VolunteerApplication = require("../models/VolunteerApplication.model.js");

// Create a new volunteer request by ORPHANAGE ADMIN

const createVolunteerRequest = asyncWrapper(async (req, res, next) => {

const { description, requiredSkills, requiredServiceType, maxVolunteers } = req.body;

const adminId = req.currentUser.id;

const orphanage = await Orphanage.findOne({ admin: adminId, status: "APPROVED" });

if (!orphanage) {

return next(appError.create("You do not have an approved orphanage", 403, httpStatusText.FAIL));

}

const newRequest = await VolunteerRequest.create({

orphanage: orphanage.\_id,

description,

requiredSkills,

requiredServiceType,

maxVolunteers: maxVolunteers || 5,

acceptedCount: 0

});

res.status(201).json({

status: httpStatusText.SUCCESS,

message: "Volunteer request created successfully.",

data: { request: newRequest }

});

});

// View all requests (for VOLUNTEERS with optional filter)

const getAllVolunteerRequests = asyncWrapper(async (req, res, next) => {

const { serviceType, limit = 4, page = 1 } = req.query;

const skip = (page - 1) \* limit;

const filter = { status: "Open" };

if (serviceType) {

filter.requiredServiceType = serviceType;

}

const requests = await VolunteerRequest.find(filter)

.skip(skip)

.limit(Number(limit))

.populate("orphanage", "name location");

res.json({

status: httpStatusText.SUCCESS,

data: { requests }

});

});

// Get request by ID (any role)

const getVolunteerRequestById = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const request = await VolunteerRequest.findById(id).populate("orphanage", "name location");

if (!request) {

return next(appError.create("Request not found", 404, httpStatusText.FAIL));

}

res.json({

status: httpStatusText.SUCCESS,

data: { request }

});

});

// Admin: Get requests for specific orphanage

const getVolunteerRequestsByOrphanageId = asyncWrapper(async (req, res, next) => {

const { orphanageId } = req.params;

const { limit = 4, page = 1 } = req.query;

const skip = (page - 1) \* limit;

const filter = {

orphanage: orphanageId,

status: "Open"

};

if (req.query.serviceType) {

filter.requiredServiceType = req.query.serviceType;

}

const requests = await VolunteerRequest.find(filter)

.skip(skip)

.limit(Number(limit))

.populate("orphanage", "name location");

res.json({

status: httpStatusText.SUCCESS,

data: { requests }

});

});

// Update request (admin only)

const updateVolunteerRequest = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

const updates = req.body;

const request = await VolunteerRequest.findById(id);

if (!request) {

return next(appError.create("Volunteer request not found", 404, httpStatusText.FAIL));

}

// تحديث الحقول المدخلة

Object.assign(request, updates);

// تحقق من تغيير العدد الأقصى للمقبولين

if (updates.maxVolunteers !== undefined) {

const acceptedCount = request.acceptedCount || 0;

if (acceptedCount < updates.maxVolunteers && request.status === "Closed") {

request.status = "Open";

}

if (acceptedCount >= updates.maxVolunteers) {

request.status = "Closed";

}

}

await request.save();

res.status(200).json({

status: httpStatusText.SUCCESS,

message: "Volunteer request updated successfully.",

data: { request }

});

});

// Delete request (admin only)

const deleteVolunteerRequest = asyncWrapper(async (req, res, next) => {

const { id } = req.params;

// تحقق من وجود الطلب أولًا

const request = await VolunteerRequest.findById(id);

if (!request) {

return next(appError.create("Volunteer request not found", 404, httpStatusText.FAIL));

}

// حذف كل التطبيقات المرتبطة بهذا الطلب

await VolunteerApplication.deleteMany({ volunteerRequest: request.\_id });

// حذف الطلب نفسه

await VolunteerRequest.findByIdAndDelete(id);

res.json({

status: httpStatusText.SUCCESS,

message: "Volunteer request and related applications deleted successfully."

});

});

const getMyVolunteerRequests = asyncWrapper(async (req, res, next) => {

const adminId = req.currentUser.id;

const orphanage = await Orphanage.findOne({ admin: adminId });

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

const filter = { orphanage: orphanage.\_id };

// Optional: allow filtering by status (e.g., Open or Closed)

if (req.query.status) {

filter.status = req.query.status;

}

const requests = await VolunteerRequest.find(filter)

.populate("orphanage", "name");

res.json({

status: httpStatusText.SUCCESS,

data: { requests }

});

});

const getOrphanageDashboardSummary = asyncWrapper(async (req, res, next) => {

const adminId = req.currentUser.id;

const orphanage = await Orphanage.findOne({ admin: adminId });

if (!orphanage) {

return next(appError.create("Orphanage not found", 404, httpStatusText.FAIL));

}

const orphanageId = orphanage.\_id;

// إجمالي الطلبات

const totalRequests = await VolunteerRequest.countDocuments({ orphanage: orphanageId });

// الطلبات المفتوحة والمغلقة

const openRequests = await VolunteerRequest.countDocuments({ orphanage: orphanageId, status: "Open" });

const closedRequests = await VolunteerRequest.countDocuments({ orphanage: orphanageId, status: "Closed" });

// جميع التطبيقات

const totalApplications = await VolunteerApplication.countDocuments({ orphanage: orphanageId });

const acceptedApplications = await VolunteerApplication.countDocuments({

orphanage: orphanageId,

status: "Accepted"

});

const rejectedApplications = await VolunteerApplication.countDocuments({

orphanage: orphanageId,

status: "Rejected"

});

const pendingApplications = await VolunteerApplication.countDocuments({

orphanage: orphanageId,

status: "Pending"

});

// توزيع حسب نوع الخدمة

const serviceTypes = ["Teaching", "Mentoring", "Healthcare", "Other"];

const requestDistributionByType = {};

const acceptedVolunteersByType = {};

for (const type of serviceTypes) {

const requestCount = await VolunteerRequest.countDocuments({

orphanage: orphanageId,

requiredServiceType: type

});

const acceptedCount = await VolunteerApplication.countDocuments({

orphanage: orphanageId,

status: "Accepted",

serviceType: type

});

requestDistributionByType[type] = requestCount;

acceptedVolunteersByType[type] = acceptedCount;

}

res.json({

status: httpStatusText.SUCCESS,

data: {

totalRequests,

openRequests,

closedRequests,

totalApplications,

acceptedApplications,

rejectedApplications,

pendingApplications,

requestDistributionByType,

acceptedVolunteersByType

}

});

});

module.exports = {

createVolunteerRequest,

getAllVolunteerRequests,

getVolunteerRequestById,

updateVolunteerRequest,

deleteVolunteerRequest,

getMyVolunteerRequests,

getOrphanageDashboardSummary

};

**File: webhook.controller.js**

const Sponsorship = require("../models/sponsorship.model.js");

const Orphan = require("../models/orphan.model.js");

const sponsorshipStatus = require("../utilities/sponsorshipStatus.js");

const stripe = require("stripe")(process.env.STRIPE\_SECRET\_KEY);

const EmergencyCampaign = require("../models/emergencyCampaign.model.js");

const Donation = require("../models/donation.model.js");

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

let event;

try {

const sig = req.headers["stripe-signature"];

event = stripe.webhooks.constructEvent(req.body, sig, process.env.STRIPE\_WEBHOOK\_SECRET);

// console.log("🔹 Full Webhook Event Payload:", JSON.stringify(event, null, 2)); // Log the full event

} catch (err) {

console.error("❌ Webhook error:", err.message);

return res.status(400).send(`Webhook error: ${err.message}`);

}

try {

switch (event.type) {

case "invoice.payment\_succeeded": {

const successInvoice = event.data.object;

console.log("🔹 Invoice Object:", successInvoice);

const updated = await Sponsorship.findOneAndUpdate(

{ subscriptionId: successInvoice.subscription },

{ status: sponsorshipStatus.ACTIVE },

{ new: true }

);

if (!updated) {

console.warn(`⚠️ No matching sponsorship found for subscription ID: ${successInvoice.subscription}`);

} else {

console.log("✅ Sponsorship updated to ACTIVE:", updated);

// Add sponsor to orphan's sponsors array

await Orphan.findByIdAndUpdate(

updated.orphan,

{ $addToSet: { sponsors: updated.sponsor } } // ensures no duplicates

);

}

break;

}

case "checkout.session.completed": {

const session = event.data.object;

// Safety: Only act if it's a donation

const donationId = session.metadata?.donationId;

console.log(donationId);

if (!donationId) {

console.warn("⚠️ No donation metadata found in checkout session.");

break;

}

const donation = await Donation.findById(donationId);

if (!donation) {

console.warn(`⚠️ Donation not found for ID ${donationId}`);

break;

}

// Avoid double updates

if (donation.status === "Completed") {

console.log(`ℹ️ Donation ${donationId} already marked as completed.`);

break;

}

donation.status = "Completed";

await donation.save();

// Update campaign currentAmount

if (donation.campaign) {

const campaign = await EmergencyCampaign.findById(donation.campaign);

if (!campaign) {

console.warn(`⚠️ Campaign not found for ID ${donation.campaign}`);

break;

}

// Update current amount

campaign.raisedAmount += donation.amount;

// Mark as completed if target reached

if (campaign.raisedAmount >= campaign.targetAmount) {

campaign.status = "Completed";

console.log(`🎉 Campaign ${campaign.\_id} reached its target and is now completed.`);

}

await campaign.save();

}

console.log(`✅ Donation ${donationId} marked as completed.`);

break;

}

case "invoice.payment\_failed": {

const failedInvoice = event.data.object;

const updated = await Sponsorship.findOneAndUpdate(

{ subscriptionId: failedInvoice.subscription },

{ status: sponsorshipStatus.FAILED }

);

if (!updated) {

console.warn(`⚠️ No matching sponsorship found for subscription ID: ${failedInvoice.subscription}`);

}

break;

}

case "customer.subscription.deleted": {

const canceledSubscription = event.data.object;

console.log("🔹 Subscription Deleted Event:", canceledSubscription.id);

// Find the sponsorship record

const sponsorship = await Sponsorship.findOne({ subscriptionId: canceledSubscription.id });

if (!sponsorship) {

console.warn(`⚠️ No matching sponsorship found for subscription ID: ${canceledSubscription.id}`);

break;

}

// Check if today's date is after the planned endDate

const today = new Date();

const endDate = new Date(sponsorship.endDate);

if (today >= endDate) {

sponsorship.status = sponsorshipStatus.COMPLETED;

console.log("✅ Sponsorship marked as COMPLETED:", sponsorship);

} else {

sponsorship.status = sponsorshipStatus.CANCELED;

console.log("❌ Sponsorship marked as CANCELED before end date:", sponsorship);

}

await sponsorship.save();

// Remove sponsor from orphan's sponsors array

await Orphan.findByIdAndUpdate(

sponsorship.orphan,

{ $pull: { sponsors: sponsorship.sponsor } }

);

break;

}

// case "customer.subscription.deleted": {

// const canceledSubscription = event.data.object;

// const updated = await Sponsorship.findOneAndUpdate(

// { subscriptionId: canceledSubscription.id },

// { status: sponsorshipStatus.CANCELED }

// );

// if (!updated) {

// console.warn(`⚠️ No matching sponsorship found for subscription ID: ${canceledSubscription.id}`);

// }

// break;

// }

default:

console.log(`ℹ️ Unhandled event type: ${event.type}`);

}

} catch (dbError) {

console.error("❌ Database update error:", dbError);

return res.status(500).send("Internal Server Error");

}

res.status(200).send("✅ Webhook received successfully");

};

module.exports = { handleWebhook };

**File: webhook.route.js**

const express = require("express");

const webhookController = require("../controllers/webhook.controller.js");

const bodyParser = require("body-parser");

const stripe = require("stripe")(process.env.STRIPE\_SECRET\_KEY);

const router = express.Router();

// Stripe will send events when:

// A payment succeeds (mark sponsorship as ACTIVE).

// A payment fails (mark sponsorship as FAILED).

// The sponsor cancels the subscription (mark as CANCELED).

// Use raw body for Stripe Webhooks

router.post(

"/stripe",

express.raw({ type: "application/json" }), // This is required for Stripe webhooks

webhookController.handleWebhook

);

module.exports = router;