// middleware/auth.js

const jwt = require('jsonwebtoken');

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

const auth = async (req, res, next) => {

try {

const token = req.header('Authorization')?.replace('Bearer ', '');

if (!token) {

return res.status(401).json({ message: 'No authentication token provided' });

}

const decoded = jwt.verify(token, process.env.JWT\_SECRET || 'your-super-secret-jwt-key-change-this-in-production');

if (!decoded.userId) {

return res.status(401).json({ message: 'Invalid token' });

}

const user = await User.findById(decoded.userId);

if (!user) {

return res.status(401).json({ message: 'User not found' });

}

req.user = {

userId: user.\_id,

name: user.name,

email: user.email

};

req.token = token;

next();

} catch (error) {

console.error('Auth middleware error:', error);

if (error.name === 'JsonWebTokenError') {

return res.status(401).json({ message: 'Invalid token' });

}

if (error.name === 'TokenExpiredError') {

return res.status(401).json({ message: 'Token expired' });

}

res.status(401).json({ message: 'Please authenticate' });

}

};

module.exports = auth;

// models/Comment.js

const mongoose = require('mongoose');

const commentSchema = new mongoose.Schema({

ticketId: { type: mongoose.Schema.Types.ObjectId, ref: 'Ticket', required: true },

userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },

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

parentId: { type: mongoose.Schema.Types.ObjectId, ref: 'Comment', default: null },

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

});

module.exports = mongoose.model('Comment', commentSchema);

// models/Project.js

const mongoose = require('mongoose');

const projectSchema = new mongoose.Schema({

name: {

type: String,

required: [true, 'Project name is required'],

trim: true

},

description: {

type: String,

required: [true, 'Project description is required']

},

status: {

type: String,

enum: ['Active', 'On Hold', 'Completed', 'Cancelled'],

default: 'Active'

},

owner: {

type: mongoose.Schema.Types.ObjectId,

ref: 'User',

required: true

},

teamMembers: [

{

name: {

type: String,

required: [true, 'Team member name is required']

},

email: {

type: String,

required: [true, 'Team member email is required'],

trim: true,

lowercase: true

}

}

],

createdAt: {

type: Date,

default: Date.now

},

updatedAt: {

type: Date,

default: Date.now

}

});

// Update the updatedAt timestamp before saving

projectSchema.pre('save', function (next) {

this.updatedAt = Date.now();

next();

});

// Remove sensitive data when converting to JSON

projectSchema.methods.toJSON = function () {

const project = this.toObject();

return project;

};

module.exports = mongoose.model('Project', projectSchema);

// models/ProjectInvitation.js

const mongoose = require('mongoose');

const projectInvitationSchema = new mongoose.Schema({

projectId: { type: mongoose.Schema.Types.ObjectId, ref: 'Project', required: true },

email: { type: String, required: true, lowercase: true, trim: true },

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

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

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

});

module.exports = mongoose.model('ProjectInvitation', projectInvitationSchema);

// models/Ticket.js

const mongoose = require('mongoose');

const commentSchema = new mongoose.Schema({

user: {

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

email: { type: String, required: true }

},

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

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

}, { \_id: true });

const ticketSchema = new mongoose.Schema({

title: {

type: String,

required: true,

trim: true

},

description: {

type: String,

required: true

},

project: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Project',

required: true

},

status: {

type: String,

enum: ['todo', 'in-progress', 'done'],

default: 'todo'

},

priority: {

type: String,

enum: ['low', 'medium', 'high', 'urgent'],

default: 'medium'

},

type: {

type: String,

enum: ['bug', 'feature', 'task'],

default: 'bug'

},

assignee: {

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

email: { type: String, required: true }

},

reporter: {

type: mongoose.Schema.Types.ObjectId,

ref: 'User',

required: true

},

comments: [{

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

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

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

}],

createdAt: {

type: Date,

default: Date.now

},

updatedAt: {

type: Date,

default: Date.now

}

});

// Update the updatedAt timestamp before saving

ticketSchema.pre('save', function (next) {

this.updatedAt = Date.now();

next();

});

module.exports = mongoose.model('Ticket', ticketSchema);

// models/User.js

const mongoose = require('mongoose');

const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({

name: {

type: String,

required: [true, 'Name is required'],

trim: true

},

email: {

type: String,

required: [true, 'Email is required'],

unique: true,

trim: true,

lowercase: true,

match: [/^[^\s@]+@[^\s@]+\.[^\s@]+$/, 'Please enter a valid email']

},

password: {

type: String,

required: [true, 'Password is required'],

minlength: [6, 'Password must be at least 6 characters long']

},

role: {

type: String,

enum: ['admin', 'manager', 'developer', 'viewer'],

default: 'developer'

},

createdAt: {

type: Date,

default: Date.now

}

});

// Hash password before saving

userSchema.pre('save', async function (next) {

try {

if (!this.isModified('password')) {

return next();

}

const salt = await bcrypt.genSalt(10);

this.password = await bcrypt.hash(this.password, salt);

next();

} catch (error) {

console.error('Password hashing error:', error);

next(error);

}

});

// Method to compare password

userSchema.methods.comparePassword = async function (candidatePassword) {

try {

return await bcrypt.compare(candidatePassword, this.password);

} catch (error) {

console.error('Password comparison error:', error);

throw error;

}

};

// Remove password when converting to JSON

userSchema.methods.toJSON = function () {

const user = this.toObject();

delete user.password;

return user;

};

module.exports = mongoose.model('User', userSchema);

// routes/auth.js

const express = require('express');

const router = express.Router();

const jwt = require('jsonwebtoken');

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

const auth = require('../middleware/auth');

// Register

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

try {

console.log('Registration request received:', { ...req.body, password: '[REDACTED]' });

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

// Validate input

if (!name || !email || !password) {

console.log('Missing required fields');

return res.status(400).json({ message: 'All fields are required' });

}

// Check if user already exists

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

if (existingUser) {

console.log('User already exists:', email);

return res.status(400).json({ message: 'User already exists' });

}

// Create new user

const user = new User({

name,

email,

password

});

await user.save();

console.log('User created successfully:', email);

// Generate token

const token = jwt.sign(

{ userId: user.\_id },

process.env.JWT\_SECRET || 'your-super-secret-jwt-key-change-this-in-production',

{ expiresIn: '7d' }

);

// Remove password from user object

const userResponse = user.toJSON();

res.status(201).json({

token,

user: userResponse

});

} catch (error) {

console.error('Registration error:', error);

if (error.name === 'ValidationError') {

return res.status(400).json({

message: 'Validation error',

errors: Object.values(error.errors).map(err => err.message)

});

}

if (error.code === 11000) {

return res.status(400).json({ message: 'Email already exists' });

}

res.status(500).json({ message: 'Error creating user' });

}

});

// Login

router.post('/login', async (req, res) => {

try {

console.log('Login request received:', { ...req.body, password: '[REDACTED]' });

const { email, password } = req.body;

// Validate input

if (!email || !password) {

console.log('Missing email or password');

return res.status(400).json({ message: 'Email and password are required' });

}

// Find user

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

if (!user) {

console.log('User not found:', email);

return res.status(401).json({ message: 'Invalid credentials' });

}

// Check password

const isMatch = await user.comparePassword(password);

if (!isMatch) {

console.log('Invalid password for user:', email);

return res.status(401).json({ message: 'Invalid credentials' });

}

console.log('User logged in successfully:', email);

// Generate token

const token = jwt.sign(

{ userId: user.\_id },

process.env.JWT\_SECRET || 'your-super-secret-jwt-key-change-this-in-production',

{ expiresIn: '7d' }

);

// Remove password from user object

const userResponse = user.toJSON();

res.json({

token,

user: userResponse

});

} catch (error) {

console.error('Login error:', error);

res.status(500).json({ message: 'Error logging in' });

}

});

// Get current user

router.get('/me', auth, async (req, res) => {

try {

const user = await User.findById(req.user.userId);

if (!user) {

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

}

res.json({ user: user.toJSON() });

} catch (error) {

console.error('Get user error:', error);

res.status(500).json({ message: 'Error fetching user data' });

}

});

module.exports = router;

// routes/comments.js

const express = require('express');

const router = express.Router();

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

const auth = require('../middleware/auth');

// Add a comment

router.post('/', auth, async (req, res) => {

try {

const { ticketId, text, parentId } = req.body;

const comment = new Comment({

ticketId,

userId: req.user.userId,

text,

parentId: parentId || null

});

await comment.save();

res.status(201).json(comment);

} catch (error) {

res.status(400).json({ message: 'Failed to add comment', error: error.message });

}

});

// Get all comments for a ticket (threaded)

router.get('/ticket/:ticketId', auth, async (req, res) => {

const comments = await Comment.find({ ticketId: req.params.ticketId }).sort({ createdAt: 1 });

res.json(comments);

});

module.exports = router;

// routes/projects.js

const express = require('express');

const router = express.Router();

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

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

const auth = require('../middleware/auth');

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

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

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

const crypto = require('crypto');

const nodemailer = require('nodemailer');

// Create project

router.post('/', auth, async (req, res) => {

try {

const { name, description, status, teamMembers } = req.body;

// Validate input

if (!name || !description) {

return res.status(400).json({ message: 'Name and description are required' });

}

const project = new Project({

name,

description,

status: status || 'Active',

owner: req.user.userId,

teamMembers: teamMembers || [] // Use teamMembers from request body or empty array

});

await project.save();

// Populate owner

await project.populate('owner', 'name email');

res.status(201).json(project);

} catch (error) {

console.error('Create project error:', error);

if (error.name === 'ValidationError') {

return res.status(400).json({ message: error.message });

}

res.status(500).json({ message: 'Error creating project' });

}

});

// Get all projects for user

router.get('/', auth, async (req, res) => {

try {

const projects = await Project.find({

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

})

.populate('owner', 'name email')

.sort({ updatedAt: -1 });

res.json(projects);

} catch (error) {

console.error('Fetch projects error:', error);

res.status(500).json({ message: 'Error fetching projects' });

}

});

// Get single project

router.get('/:id', auth, async (req, res) => {

try {

const project = await Project.findOne({

\_id: req.params.id,

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

})

.populate('owner', 'name email');

if (!project) {

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

}

res.json(project);

} catch (error) {

console.error('Fetch project error:', error);

res.status(500).json({ message: 'Error fetching project' });

}

});

// Update project

router.put('/:id', auth, async (req, res) => {

try {

const { name, description, status } = req.body;

const project = await Project.findOne({

\_id: req.params.id,

owner: req.user.userId

});

if (!project) {

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

}

project.name = name;

project.description = description;

project.status = status;

await project.save();

res.json(project);

} catch (error) {

console.error('Update project error:', error);

res.status(500).json({ message: 'Error updating project' });

}

});

// Delete project

router.delete('/:id', auth, async (req, res) => {

try {

const project = await Project.findOne({

\_id: req.params.id,

owner: req.user.userId

});

if (!project) {

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

}

// Find all tickets in this project

const tickets = await Ticket.find({ project: project.\_id });

// Delete all comments for these tickets

const ticketIds = tickets.map(t => t.\_id);

await Comment.deleteMany({ ticketId: { $in: ticketIds } });

// Delete all tickets

await Ticket.deleteMany({ project: project.\_id });

// Delete the project

await project.deleteOne();

res.json({ message: 'Project, tickets, and related comments deleted' });

} catch (error) {

console.error('Delete project error:', error);

res.status(500).json({ message: 'Error deleting project' });

}

});

// Add team member

router.post('/:id/members', auth, async (req, res) => {

try {

const { name, email } = req.body;

const project = await Project.findOne({

\_id: req.params.id,

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

});

if (!project) {

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

}

// Check if member already exists

const isMember = project.teamMembers.some(member => member.email === email);

if (isMember) {

return res.status(400).json({ message: 'Team member already exists' });

}

project.teamMembers.push({ name, email });

await project.save();

res.json(project);

} catch (error) {

console.error('Add team member error:', error);

res.status(500).json({ message: 'Error adding team member' });

}

});

// Remove team member

router.delete('/:id/members/:email', auth, async (req, res) => {

try {

const project = await Project.findOne({

\_id: req.params.id,

owner: req.user.userId

});

if (!project) {

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

}

project.teamMembers = project.teamMembers.filter(

member => member.email !== req.params.email

);

await project.save();

res.json(project);

} catch (error) {

console.error('Remove team member error:', error);

res.status(500).json({ message: 'Error removing team member' });

}

});

// Helper: send invitation email

async function sendInvitationEmail(email, project, token) {

// Configure your email service here (Gmail example)

const transporter = nodemailer.createTransport({

service: 'gmail',

auth: {

user: process.env.EMAIL\_USER, // set in .env

pass: process.env.EMAIL\_PASS // set in .env

}

});

const acceptUrl = `http://localhost:3000/accept-invite/${token}`;

await transporter.sendMail({

from: process.env.EMAIL\_USER,

to: email,

subject: `Invitation to join project: ${project.name}`,

text: `You have been invited to join the project "${project.name}".\nDescription: ${project.description}\n\nClick here to accept: ${acceptUrl}`

});

}

// Invite member (admin only)

router.post('/:id/invite', auth, async (req, res) => {

try {

const { email } = req.body;

const project = await Project.findOne({

\_id: req.params.id,

owner: req.user.userId

});

if (!project) {

return res.status(403).json({ message: 'Only the project owner can invite members.' });

}

// Check if already a member

if (project.teamMembers.some(m => m.email === email)) {

return res.status(400).json({ message: 'User is already a team member.' });

}

// Check for existing pending invite

let invite = await ProjectInvitation.findOne({ projectId: project.\_id, email, accepted: false });

if (invite) {

return res.status(400).json({ message: 'Invitation already sent.' });

}

// Create invite

const token = crypto.randomBytes(24).toString('hex');

invite = new ProjectInvitation({ projectId: project.\_id, email, token });

await invite.save();

// Send email

await sendInvitationEmail(email, project, token);

res.json({ message: 'Invitation sent.' });

} catch (error) {

console.error('Invite error:', error);

res.status(500).json({ message: 'Error sending invitation.' });

}

});

// Accept invite (user must be logged in and email must match)

router.post('/accept-invite/:token', auth, async (req, res) => {

try {

const invite = await ProjectInvitation.findOne({ token: req.params.token, accepted: false });

if (!invite) {

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

}

// Only allow if logged in user email matches invite email

const user = await User.findById(req.user.userId);

if (!user || user.email !== invite.email) {

return res.status(403).json({ message: 'You must be logged in with the invited email to accept.' });

}

// Add to project teamMembers if not already

const project = await Project.findById(invite.projectId);

if (!project.teamMembers.some(m => m.email === user.email)) {

project.teamMembers.push({ name: user.name, email: user.email });

await project.save();

}

invite.accepted = true;

await invite.save();

res.json({ message: 'You have joined the project.' });

} catch (error) {

console.error('Accept invite error:', error);

res.status(500).json({ message: 'Error accepting invitation.' });

}

});

module.exports = router;

// routes/tickets.js

const express = require('express');

const router = express.Router();

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

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

const auth = require('../middleware/auth');

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

// Create ticket

router.post('/', auth, async (req, res) => {

try {

const { title, description, project, priority, assignee, comment } = req.body;

// Check if user has access to the project

const projectDoc = await Project.findOne({

\_id: project,

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

});

if (!projectDoc) {

return res.status(403).json({ message: 'Access denied to project' });

}

// Validate assignee is a team member

const isTeamMember = projectDoc.teamMembers.some(

(tm) => tm.email === assignee?.email && tm.name === assignee?.name

);

if (!isTeamMember) {

return res.status(400).json({ message: 'Assignee must be a project team member' });

}

const ticketData = {

title,

description,

project,

priority,

assignee,

reporter: req.user.userId,

status: req.body.status || 'todo'

};

if (comment && comment.trim()) {

ticketData.comments = [{ user: req.user.userId, text: comment.trim() }];

}

const ticket = new Ticket(ticketData);

await ticket.save();

res.status(201).json(ticket);

} catch (error) {

res.status(500).json({ message: 'Error creating ticket' });

}

});

// Get tickets for a project with AND filters and OR for keyword

router.get('/project/:projectId', auth, async (req, res) => {

try {

const { status, priority, assignee, q } = req.query;

// Check if user has access to the project

const project = await Project.findOne({

\_id: req.params.projectId,

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

});

if (!project) {

return res.status(403).json({ message: 'Access denied to project' });

}

// Build AND filter

const filter = { project: req.params.projectId };

if (status) filter.status = status;

if (priority) filter.priority = priority;

if (assignee) filter['assignee.name'] = assignee;

if (q) filter.$or = [

{ title: { $regex: q, $options: 'i' } },

{ description: { $regex: q, $options: 'i' } }

];

const tickets = await Ticket.find(filter);

res.json(tickets);

} catch (error) {

res.status(500).json({ message: 'Error fetching tickets' });

}

});

// Get single ticket

router.get('/:id', auth, async (req, res) => {

try {

const ticket = await Ticket.findById(req.params.id);

if (!ticket) {

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

}

// Check if user has access to the project

const project = await Project.findOne({

\_id: ticket.project,

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

});

if (!project) {

return res.status(403).json({ message: 'Access denied to ticket' });

}

res.json(ticket);

} catch (error) {

res.status(500).json({ message: 'Error fetching ticket' });

}

});

// Update ticket

router.put('/:id', auth, async (req, res) => {

try {

const ticket = await Ticket.findById(req.params.id);

if (!ticket) {

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

}

// Check if user has access to the project

const project = await Project.findOne({

\_id: ticket.project,

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

});

if (!project) {

return res.status(403).json({ message: 'Access denied to ticket' });

}

// Only allow specific fields to be updated

const allowedUpdates = ['title', 'description', 'status', 'priority', 'assignee'];

allowedUpdates.forEach(field => {

if (req.body[field] !== undefined) {

ticket[field] = req.body[field];

}

});

// If updating assignee, validate

if (req.body.assignee) {

const isTeamMember = project.teamMembers.some(

(tm) => tm.email === req.body.assignee.email && tm.name === req.body.assignee.name

);

if (!isTeamMember) {

return res.status(400).json({ message: 'Assignee must be a project team member' });

}

}

await ticket.save();

res.json(ticket);

} catch (error) {

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

}

});

// Delete ticket

router.delete('/:id', auth, async (req, res) => {

try {

const ticket = await Ticket.findById(req.params.id);

if (!ticket) {

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

}

// Check if user has access to the project

const project = await Project.findOne({

\_id: ticket.project,

$or: [

{ owner: req.user.userId },

{ 'teamMembers.email': req.user.email }

]

});

if (!project) {

return res.status(403).json({ message: 'Access denied to ticket' });

}

// Delete all comments for this ticket

await Comment.deleteMany({ ticketId: ticket.\_id });

await ticket.deleteOne();

res.json({ message: 'Ticket and related comments deleted' });

} catch (error) {

console.error('Error deleting ticket:', error);

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

}

});

// Add comment to ticket

router.post('/:id/comments', auth, async (req, res) => {

try {

const ticket = await Ticket.findById(req.params.id);

if (!ticket) {

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

}

if (!req.body.text || !req.body.text.trim()) {

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

}

// Log user info for debugging

console.log('Adding comment as user:', req.user);

ticket.comments.push({

user: {

name: req.user.name,

email: req.user.email

},

text: req.body.text.trim()

});

await ticket.save();

res.json(ticket.comments);

} catch (error) {

console.error('Error adding comment:', error);

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

}

});

// Get all comments for a ticket

router.get('/:id/comments', auth, async (req, res) => {

try {

const ticket = await Ticket.findById(req.params.id);

if (!ticket) {

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

}

res.json(ticket.comments);

} catch (error) {

res.status(500).json({ message: 'Error fetching comments' });

}

});

module.exports = router;

//backend/app.js

app.use('/api/comments', require('./routes/comments'));

// server.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const helmet = require('helmet');

const dotenv = require('dotenv');

// Load environment variables

dotenv.config();

// Create Express app

const app = express();

// Middleware

app.use(helmet());

app.use(cors());

app.use(express.json());

// MongoDB Connection

mongoose.connect(process.env.MONGODB\_URI || 'mongodb://localhost:27017/bug-tracker', {

useNewUrlParser: true,

useUnifiedTopology: true,

})

.then(() => console.log('MongoDB Connected'))

.catch(err => console.error('MongoDB Connection Error:', err));

// Routes

app.use('/api/auth', require('./routes/auth'));

app.use('/api/projects', require('./routes/projects'));

app.use('/api/tickets', require('./routes/tickets'));

app.use('/api/comments', require('./routes/comments'));

// Error handling middleware

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

console.error(err.stack);

res.status(500).json({ message: 'Something went wrong!' });

});

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

app.listen(PORT, () => {

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

});

<!-- public/index.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<meta name="theme-color" content="#000000" />

<meta name="description" content="Bug Tracker - Track and manage your software bugs efficiently" />

<link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" />

<link rel="manifest" href="%PUBLIC\_URL%/manifest.json" />

<script src="https://cdn.tailwindcss.com"></script>

<title>Bug Tracker</title>

</head>

<body>

<noscript>You need to enable JavaScript to run this app.</noscript>

<div id="root"></div>

</body>

</html>

// src/components/KanbanBoard.js

import React, { useState, useEffect } from 'react';

import { DragDropContext, Droppable, Draggable } from '@hello-pangea/dnd';

import axios from 'axios';

// StrictMode compatible wrapper

const StrictModeDroppable = ({ children, ...props }) => {

const [enabled, setEnabled] = useState(false);

useEffect(() => {

const animation = requestAnimationFrame(() => setEnabled(true));

return () => {

cancelAnimationFrame(animation);

setEnabled(false);

};

}, []);

if (!enabled) {

return null;

}

return <Droppable {...props}>{children}</Droppable>;

};

const KanbanBoard = ({ projectId }) => {

const [tickets, setTickets] = useState({

'todo': [],

'in-progress': [],

'done': []

});

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

useEffect(() => {

fetchTickets();

}, [projectId]);

const fetchTickets = async () => {

try {

setLoading(true);

const response = await axios.get(`http://localhost:5000/api/tickets/project/${projectId}`, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

// Validate and group tickets

const groupedTickets = response.data.reduce((acc, ticket) => {

// Validate ticket data

if (!ticket || !ticket.\_id) {

console.warn('Invalid ticket data:', ticket);

return acc;

}

// Normalize status

let status = (ticket.status || 'todo').toLowerCase().replace(/\s+/g, '-');

if (!['todo', 'in-progress', 'done'].includes(status)) {

status = 'todo';

}

// Ensure the ticket has all required fields

const normalizedTicket = {

\_id: ticket.\_id.toString(), // Convert ObjectId to string

title: ticket.title || 'Untitled',

description: ticket.description || '',

priority: ticket.priority || 'medium',

status: status,

assignee: ticket.assignee || null,

projectId: ticket.project || projectId

};

if (!acc[status]) acc[status] = [];

acc[status].push(normalizedTicket);

return acc;

}, {

'todo': [],

'in-progress': [],

'done': []

});

setTickets(groupedTickets);

setError(null);

} catch (error) {

console.error('Error fetching tickets:', error);

setError('Failed to load tickets');

} finally {

setLoading(false);

}

};

const onDragEnd = async (result) => {

const { source, destination, draggableId } = result;

// Validate drag result

if (!destination || !draggableId) return;

// If dropped in the same place

if (

source.droppableId === destination.droppableId &&

source.index === destination.index

) return;

try {

// Create new ticket arrays

const sourceColumn = [...tickets[source.droppableId]];

const destColumn = source.droppableId === destination.droppableId

? sourceColumn

: [...tickets[destination.droppableId]];

// Find the ticket being moved

const ticketIndex = sourceColumn.findIndex(ticket => ticket.\_id === draggableId);

if (ticketIndex === -1) {

throw new Error(`Ticket with id ${draggableId} not found`);

}

// Remove ticket from source

const [movedTicket] = sourceColumn.splice(ticketIndex, 1);

// Update ticket status

movedTicket.status = destination.droppableId;

// Add ticket to destination

destColumn.splice(destination.index, 0, movedTicket);

// Update state

const newTickets = {

...tickets,

[source.droppableId]: sourceColumn,

[destination.droppableId]: destColumn

};

setTickets(newTickets);

// Update ticket status in backend

await axios.patch(

`http://localhost:5000/api/tickets/${draggableId}`,

{ status: destination.droppableId },

{

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

}

);

} catch (error) {

console.error('Error updating ticket:', error);

// Only re-fetch if the backend update fails

fetchTickets();

}

};

const columns = {

'todo': { title: 'To Do', color: 'bg-gray-100' },

'in-progress': { title: 'In Progress', color: 'bg-blue-100' },

'done': { title: 'Done', color: 'bg-green-100' }

};

if (loading) {

return <div className="text-center p-4">Loading tickets...</div>;

}

if (error) {

return <div className="text-center text-red-600 p-4">{error}</div>;

}

return (

<DragDropContext onDragEnd={onDragEnd}>

<div className="grid grid-cols-1 md:grid-cols-3 gap-4 p-4">

{Object.entries(columns).map(([columnId, column]) => (

<div key={columnId} className="flex flex-col">

<h3 className="text-lg font-semibold mb-4">

{column.title} ({tickets[columnId]?.length || 0})

</h3>

<StrictModeDroppable droppableId={columnId}>

{(provided, snapshot) => (

<div

ref={provided.innerRef}

{...provided.droppableProps}

className={`${column.color} p-4 rounded-lg min-h-[500px] ${snapshot.isDraggingOver ? 'ring-2 ring-blue-500' : ''

}`}

>

{tickets[columnId].map((ticket, index) => (

<Draggable

key={ticket.\_id}

draggableId={ticket.\_id}

index={index}

>

{(provided, snapshot) => (

<div

ref={provided.innerRef}

{...provided.draggableProps}

{...provided.dragHandleProps}

className={`bg-white p-4 mb-4 rounded shadow ${snapshot.isDragging ? 'ring-2 ring-blue-500' : ''

}`}

>

<h4 className="font-medium">{ticket.title}</h4>

<p className="text-sm text-gray-600 mt-2">

{ticket.description}

</p>

<div className="mt-2 flex items-center justify-between">

<span className={`px-2 py-1 rounded text-xs ${ticket.priority === 'high' ? 'bg-red-100 text-red-800' :

ticket.priority === 'medium' ? 'bg-yellow-100 text-yellow-800' :

'bg-green-100 text-green-800'

}`}>

{ticket.priority}

</span>

{ticket.assignee && (

<span className="text-sm text-gray-500">

{ticket.assignee.name}

</span>

)}

</div>

</div>

)}

</Draggable>

))}

{provided.placeholder}

</div>

)}

</StrictModeDroppable>

</div>

))}

</div>

</DragDropContext>

);

};

export default KanbanBoard;

// src/components/PrivateRoute.js

import React from 'react';

import { Navigate } from 'react-router-dom';

import { useAuth } from '../contexts/AuthContext';

const PrivateRoute = ({ children }) => {

const { currentUser, loading } = useAuth();

if (loading) {

return (

<div className="min-h-screen flex items-center justify-center">

<div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-primary-600"></div>

</div>

);

}

return currentUser ? children : <Navigate to="/login" />;

};

export default PrivateRoute;

// src/components/Sidebar.js

import React from 'react';

import { Link, useLocation } from 'react-router-dom';

const Sidebar = ({ onLogout }) => {

const location = useLocation();

return (

<aside className="bg-white w-64 min-h-screen border-r hidden md:block">

<div className="p-6">

<h2 className="text-2xl font-bold mb-8 text-blue-700">Bug Tracker</h2>

<nav className="flex flex-col gap-4">

<Link to="/dashboard" className={`py-2 px-4 rounded hover:bg-blue-50 ${location.pathname === '/dashboard' ? 'bg-blue-100 text-blue-700' : 'text-gray-700'}`}>Dashboard</Link>

<Link to="/projects/new" className={`py-2 px-4 rounded hover:bg-blue-50 ${location.pathname === '/projects/new' ? 'bg-blue-100 text-blue-700' : 'text-gray-700'}`}>Create Project</Link>

<Link to="/projects" className={`py-2 px-4 rounded hover:bg-blue-50 ${location.pathname.startsWith('/projects') && location.pathname !== '/projects/new' ? 'bg-blue-100 text-blue-700' : 'text-gray-700'}`}>Projects</Link>

<button onClick={onLogout} className="mt-8 py-2 px-4 rounded bg-red-600 text-white hover:bg-red-700">Logout</button>

</nav>

</div>

</aside>

);

};

export default Sidebar;

// src/contexts/AuthContext.js

import React, { createContext, useState, useContext, useEffect } from 'react';

import axios from 'axios';

const AuthContext = createContext(null);

export const useAuth = () => {

const context = useContext(AuthContext);

if (!context) {

throw new Error('useAuth must be used within an AuthProvider');

}

return context;

};

export const AuthProvider = ({ children }) => {

const [currentUser, setCurrentUser] = useState(null);

const [loading, setLoading] = useState(true);

useEffect(() => {

const token = localStorage.getItem('token');

if (token) {

axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;

checkAuth();

} else {

setLoading(false);

}

}, []);

const checkAuth = async () => {

try {

const response = await axios.get('http://localhost:5000/api/auth/me');

setCurrentUser(response.data.user);

} catch (error) {

console.error('Auth check failed:', error);

localStorage.removeItem('token');

delete axios.defaults.headers.common['Authorization'];

setCurrentUser(null);

} finally {

setLoading(false);

}

};

const login = async (email, password) => {

try {

const response = await axios.post('http://localhost:5000/api/auth/login', {

email,

password,

});

const { token, user } = response.data;

localStorage.setItem('token', token);

axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;

setCurrentUser(user);

return user;

} catch (error) {

console.error('Login failed:', error);

throw new Error(error.response?.data?.message || 'Failed to login');

}

};

const register = async (email, password, name) => {

try {

const response = await axios.post('http://localhost:5000/api/auth/register', {

name,

email,

password,

});

const { token, user } = response.data;

localStorage.setItem('token', token);

axios.defaults.headers.common['Authorization'] = `Bearer ${token}`;

setCurrentUser(user);

return user;

} catch (error) {

console.error('Registration failed:', error);

throw new Error(error.response?.data?.message || 'Failed to register');

}

};

const logout = () => {

localStorage.removeItem('token');

delete axios.defaults.headers.common['Authorization'];

setCurrentUser(null);

};

const value = {

currentUser,

loading,

login,

register,

logout,

};

return (

<AuthContext.Provider value={value}>

{!loading && children}

</AuthContext.Provider>

);

};

// src/pages/CreateProject.js

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import axios from 'axios';

const CreateProject = () => {

const [name, setName] = useState('');

const [description, setDescription] = useState('');

const [teamMembers, setTeamMembers] = useState([]);

const [loading, setLoading] = useState(false);

const [error, setError] = useState('');

const navigate = useNavigate();

const handleTeamMemberChange = (index, field, value) => {

const updated = [...teamMembers];

updated[index][field] = value;

setTeamMembers(updated);

};

const handleAddTeamMember = () => {

if (

teamMembers.length === 0 ||

(teamMembers[teamMembers.length - 1].name.trim() && teamMembers[teamMembers.length - 1].email.trim())

) {

setTeamMembers([...teamMembers, { name: '', email: '' }]);

}

};

const handleRemoveTeamMember = (index) => {

setTeamMembers(teamMembers.filter((\_, i) => i !== index));

};

const handleSubmit = async (e) => {

e.preventDefault();

setError('');

// Validate team members

const hasIncomplete = teamMembers.some(

(tm) => (tm.name.trim() && !tm.email.trim()) || (!tm.name.trim() && tm.email.trim())

);

if (hasIncomplete) {

setError('Please fill out both name and email for all team members or remove incomplete rows.');

return;

}

const filteredMembers = teamMembers

.map((tm) => ({ name: tm.name.trim(), email: tm.email.trim() }))

.filter((tm) => tm.name && tm.email);

try {

setLoading(true);

const response = await axios.post(

'http://localhost:5000/api/projects',

{

name,

description,

status: 'Active',

teamMembers: filteredMembers

},

{

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

}

);

navigate(`/projects/${response.data.\_id}`);

} catch (error) {

setError(

error.response?.data?.message || 'Failed to create project. Please try again.'

);

} finally {

setLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-100 py-12 px-4 sm:px-6 lg:px-8">

<div className="max-w-2xl mx-auto">

<div className="bg-white shadow sm:rounded-lg">

<div className="px-4 py-5 sm:p-6">

<h3 className="text-2xl leading-6 font-bold text-gray-900 mb-2">Create New Project</h3>

<p className="text-gray-500 mb-6">Fill in the details and add your team members.</p>

{error && (

<div className="mt-4 bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

)}

<form onSubmit={handleSubmit} className="mt-6 space-y-6">

<div>

<label htmlFor="name" className="block text-sm font-medium text-gray-700">

Project Name

</label>

<div className="mt-1">

<input

type="text"

name="name"

id="name"

required

value={name}

onChange={(e) => setName(e.target.value)}

className="shadow-sm focus:ring-blue-500 focus:border-blue-500 block w-full sm:text-sm border-gray-300 rounded-md"

placeholder="Enter project name"

/>

</div>

</div>

<div>

<label htmlFor="description" className="block text-sm font-medium text-gray-700">

Description

</label>

<div className="mt-1">

<textarea

id="description"

name="description"

rows={4}

required

value={description}

onChange={(e) => setDescription(e.target.value)}

className="shadow-sm focus:ring-blue-500 focus:border-blue-500 block w-full sm:text-sm border-gray-300 rounded-md"

placeholder="Enter project description"

/>

</div>

</div>

<div>

<label className="block text-sm font-medium text-gray-700 mb-2">

Team Members

</label>

<div className="space-y-4">

{teamMembers.map((member, idx) => (

<div key={idx} className="flex flex-col sm:flex-row sm:space-x-4 items-center bg-gray-50 p-3 rounded-md border border-gray-200">

<input

type="text"

placeholder="Name"

value={member.name}

onChange={(e) => handleTeamMemberChange(idx, 'name', e.target.value)}

className="mb-2 sm:mb-0 flex-1 px-3 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-blue-500 focus:border-blue-500"

/>

<input

type="email"

placeholder="Email"

value={member.email}

onChange={(e) => handleTeamMemberChange(idx, 'email', e.target.value)}

className="mb-2 sm:mb-0 flex-1 px-3 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-blue-500 focus:border-blue-500"

/>

<button

type="button"

onClick={() => handleRemoveTeamMember(idx)}

className="ml-0 sm:ml-2 px-3 py-2 bg-red-500 text-white rounded-md hover:bg-red-600 focus:outline-none"

>

Remove

</button>

</div>

))}

</div>

<button

type="button"

onClick={handleAddTeamMember}

className="mt-3 px-4 py-2 bg-blue-100 text-blue-700 rounded-md hover:bg-blue-200 focus:outline-none"

>

+ Add Team Member

</button>

</div>

<div className="flex justify-end space-x-3">

<button

type="button"

onClick={() => navigate('/')}

className="bg-white py-2 px-4 border border-gray-300 rounded-md shadow-sm text-sm font-medium text-gray-700 hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500"

>

Cancel

</button>

<button

type="submit"

disabled={loading}

className="inline-flex justify-center py-2 px-4 border border-transparent shadow-sm text-sm font-medium rounded-md text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50"

>

{loading ? 'Creating...' : 'Create Project'}

</button>

</div>

</form>

</div>

</div>

</div>

</div>

);

};

export default CreateProject;

// src/pages/CreateTicket.js

import React, { useState, useEffect } from 'react';

import { useParams, useNavigate } from 'react-router-dom';

import axios from 'axios';

const CreateTicket = () => {

const { projectId } = useParams();

const [title, setTitle] = useState('');

const [description, setDescription] = useState('');

const [priority, setPriority] = useState('medium');

const [assignee, setAssignee] = useState('');

const [teamMembers, setTeamMembers] = useState([]);

const [loading, setLoading] = useState(false);

const [error, setError] = useState('');

const [comment, setComment] = useState('');

const navigate = useNavigate();

useEffect(() => {

// Fetch project to get team members

const fetchProject = async () => {

try {

const response = await axios.get(`http://localhost:5000/api/projects/${projectId}`);

setTeamMembers(response.data.teamMembers || []);

} catch (err) {

setError('Failed to fetch project team members');

}

};

fetchProject();

}, [projectId]);

const handleSubmit = async (e) => {

e.preventDefault();

setError('');

if (!assignee) {

setError('Please select an assignee.');

return;

}

const selectedAssignee = teamMembers.find(tm => tm.email === assignee);

if (!selectedAssignee) {

setError('Invalid assignee selected.');

return;

}

try {

setLoading(true);

await axios.post('http://localhost:5000/api/tickets', {

title,

description,

priority,

assignee: selectedAssignee,

project: projectId,

comment: comment.trim() ? comment : undefined

});

navigate(`/projects/${projectId}`);

} catch (error) {

setError('Failed to create ticket. Please try again.');

} finally {

setLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-50 py-12 px-4 sm:px-6 lg:px-8">

<div className="max-w-3xl mx-auto">

<div className="md:flex md:items-center md:justify-between">

<div className="flex-1 min-w-0">

<h2 className="text-2xl font-bold leading-7 text-gray-900 sm:text-3xl sm:truncate">

Create New Ticket

</h2>

</div>

</div>

<div className="mt-8 bg-white shadow sm:rounded-lg">

<div className="px-4 py-5 sm:p-6">

{error && (

<div className="mb-4 bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

)}

<form onSubmit={handleSubmit} className="space-y-6">

<div>

<label htmlFor="title" className="block text-sm font-medium text-gray-700">

Title

</label>

<div className="mt-1">

<input

type="text"

name="title"

id="title"

required

value={title}

onChange={(e) => setTitle(e.target.value)}

className="shadow-sm focus:ring-primary-500 focus:border-primary-500 block w-full sm:text-sm border-gray-300 rounded-md"

/>

</div>

</div>

<div>

<label htmlFor="description" className="block text-sm font-medium text-gray-700">

Description

</label>

<div className="mt-1">

<textarea

id="description"

name="description"

rows={4}

required

value={description}

onChange={(e) => setDescription(e.target.value)}

className="shadow-sm focus:ring-primary-500 focus:border-primary-500 block w-full sm:text-sm border-gray-300 rounded-md"

/>

</div>

</div>

<div>

<label htmlFor="comment" className="block text-sm font-medium text-gray-700">

Initial Comment (optional)

</label>

<div className="mt-1">

<textarea

id="comment"

name="comment"

rows={3}

value={comment}

onChange={(e) => setComment(e.target.value)}

className="shadow-sm focus:ring-primary-500 focus:border-primary-500 block w-full sm:text-sm border-gray-300 rounded-md"

placeholder="Add an initial comment..."

/>

</div>

</div>

<div className="grid grid-cols-1 gap-6 sm:grid-cols-2">

<div>

<label htmlFor="priority" className="block text-sm font-medium text-gray-700">

Priority

</label>

<select

id="priority"

name="priority"

value={priority}

onChange={(e) => setPriority(e.target.value)}

className="mt-1 block w-full pl-3 pr-10 py-2 text-base border-gray-300 focus:outline-none focus:ring-primary-500 focus:border-primary-500 sm:text-sm rounded-md"

>

<option value="low">Low</option>

<option value="medium">Medium</option>

<option value="high">High</option>

<option value="urgent">Urgent</option>

</select>

</div>

<div>

<label htmlFor="assignee" className="block text-sm font-medium text-gray-700">

Assignee

</label>

<select

id="assignee"

name="assignee"

value={assignee}

onChange={(e) => setAssignee(e.target.value)}

required

className="mt-1 block w-full pl-3 pr-10 py-2 text-base border-gray-300 focus:outline-none focus:ring-primary-500 focus:border-primary-500 sm:text-sm rounded-md"

>

<option value="">Select team member</option>

{teamMembers.map((tm) => (

<option key={tm.email} value={tm.email}>

{tm.name} ({tm.email})

</option>

))}

</select>

</div>

</div>

<div className="flex justify-end space-x-3">

<button

type="button"

onClick={() => navigate(`/projects/${projectId}`)}

className="inline-flex items-center px-4 py-2 border border-gray-300 shadow-sm text-sm font-medium rounded-md text-gray-700 bg-white hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary-500"

>

Cancel

</button>

<button

type="submit"

disabled={loading}

className="inline-flex items-center px-4 py-2 border border-transparent text-sm font-medium rounded-md text-white bg-primary-600 hover:bg-primary-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary-500 disabled:opacity-50"

>

{loading ? 'Creating...' : 'Create Ticket'}

</button>

</div>

</form>

</div>

</div>

</div>

</div>

);

};

export default CreateTicket;

// src/pages/Dashboard.js

import React, { useState, useEffect } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useAuth } from '../contexts/AuthContext';

import axios from 'axios';

import Sidebar from '../components/Sidebar';

import Modal from 'react-modal';

const Dashboard = () => {

const [projects, setProjects] = useState([]);

const [selectedProject, setSelectedProject] = useState('');

const [loading, setLoading] = useState(true);

const [error, setError] = useState('');

const { currentUser, logout } = useAuth();

const navigate = useNavigate();

// New state variables for project editing

const [editModalOpen, setEditModalOpen] = useState(false);

const [deleteModalOpen, setDeleteModalOpen] = useState(false);

const [selectedProjectForEdit, setSelectedProjectForEdit] = useState(null);

const [editFields, setEditFields] = useState({ name: '', description: '', status: '' });

const [editTeamMembers, setEditTeamMembers] = useState([]);

const [newTeamMember, setNewTeamMember] = useState({ name: '', email: '' });

useEffect(() => {

fetchProjects();

}, []);

const fetchProjects = async () => {

try {

const response = await axios.get('http://localhost:5000/api/projects', {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

setProjects(response.data);

setLoading(false);

} catch (error) {

setError('Failed to fetch projects');

setLoading(false);

}

};

const handleLogout = async () => {

try {

await logout();

} catch (error) {

setError('Failed to log out');

}

};

const handleProjectSelect = (e) => {

setSelectedProject(e.target.value);

if (e.target.value) {

navigate(`/projects/${e.target.value}`);

}

};

const handleEditClick = (e, project) => {

e.preventDefault();

e.stopPropagation();

setSelectedProjectForEdit(project);

setEditFields({

name: project.name,

description: project.description,

status: project.status

});

setEditTeamMembers(project.teamMembers || []);

setEditModalOpen(true);

};

const handleDeleteClick = (e, project) => {

e.preventDefault(); // Prevent navigation

e.stopPropagation(); // Prevent event bubbling

setSelectedProjectForEdit(project);

setDeleteModalOpen(true);

};

const handleEditSubmit = async (e) => {

e.preventDefault();

try {

await axios.put(`http://localhost:5000/api/projects/${selectedProjectForEdit.\_id}`, editFields);

setEditModalOpen(false);

fetchProjects();

} catch (error) {

setError('Failed to update project');

}

};

const handleDeleteConfirm = async () => {

try {

await axios.delete(`http://localhost:5000/api/projects/${selectedProjectForEdit.\_id}`);

setDeleteModalOpen(false);

fetchProjects();

} catch (error) {

setError('Failed to delete project');

}

};

const handleAddTeamMember = async (e) => {

e.preventDefault();

if (!newTeamMember.name.trim() || !newTeamMember.email.trim()) return;

try {

await axios.post(`http://localhost:5000/api/projects/${selectedProjectForEdit.\_id}/members`, newTeamMember);

setNewTeamMember({ name: '', email: '' });

// Refresh team members

const { data } = await axios.get(`http://localhost:5000/api/projects/${selectedProjectForEdit.\_id}`);

setEditTeamMembers(data.teamMembers || []);

fetchProjects();

} catch (error) {

setError('Failed to add team member');

}

};

const handleRemoveTeamMember = async (email) => {

try {

await axios.delete(`http://localhost:5000/api/projects/${selectedProjectForEdit.\_id}/members/${email}`);

// Refresh team members

const { data } = await axios.get(`http://localhost:5000/api/projects/${selectedProjectForEdit.\_id}`);

setEditTeamMembers(data.teamMembers || []);

fetchProjects();

} catch (error) {

setError('Failed to remove team member');

}

};

if (loading) {

return (

<div className="min-h-screen bg-gray-100 p-6">

<div className="max-w-7xl mx-auto">

<div className="text-center">Loading...</div>

</div>

</div>

);

}

return (

<div className="min-h-screen bg-gray-100 flex">

<Sidebar onLogout={handleLogout} />

<div className="flex-1 flex flex-col">

{/\* Topbar for mobile \*/}

<nav className="md:hidden bg-white shadow px-4 py-3 flex justify-between items-center">

<h1 className="text-xl font-bold text-gray-900">Bug Tracker</h1>

<button onClick={handleLogout} className="bg-red-600 text-white px-4 py-2 rounded-md text-sm font-medium hover:bg-red-700">Logout</button>

</nav>

<main className="flex-1 p-6">

{/\* Breadcrumbs \*/}

<nav className="text-sm mb-4" aria-label="Breadcrumb">

<ol className="list-reset flex text-gray-500">

<li><Link to="/dashboard" className="hover:underline text-blue-600">Dashboard</Link></li>

{selectedProject && (

<>

<li><span className="mx-2">/</span></li>

<li className="text-gray-700 font-medium">{projects.find(p => p.\_id === selectedProject)?.name}</li>

</>

)}

</ol>

</nav>

{/\* Project Selector \*/}

<div className="flex flex-col sm:flex-row sm:items-center sm:justify-between mb-6 gap-4">

<h2 className="text-2xl font-bold text-gray-900">Your Projects</h2>

<div className="flex gap-2 items-center">

<select

value={selectedProject}

onChange={handleProjectSelect}

className="border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

>

<option value="">Select Project</option>

{projects.map((project) => (

<option key={project.\_id} value={project.\_id}>{project.name}</option>

))}

</select>

<Link

to="/projects/new"

className="bg-blue-600 text-white px-4 py-2 rounded-md text-sm font-medium hover:bg-blue-700"

>

Create New Project

</Link>

</div>

</div>

{error && (

<div className="mb-4 bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

)}

{projects.length === 0 ? (

<div className="text-center py-12">

<h3 className="text-lg font-medium text-gray-900">No projects yet</h3>

<p className="mt-1 text-sm text-gray-500">Get started by creating a new project.</p>

</div>

) : (

<div className="grid grid-cols-1 gap-4 sm:grid-cols-2 lg:grid-cols-3">

{projects.map((project) => (

<div

key={project.\_id}

className="block bg-white overflow-hidden shadow rounded-lg hover:shadow-md transition-shadow duration-200"

>

<div className="px-4 py-5 sm:p-6">

<div className="flex justify-between items-start">

<Link

to={`/projects/${project.\_id}`}

className="flex-1"

>

<h3 className="text-lg font-medium text-gray-900">{project.name}</h3>

<p className="mt-1 text-sm text-gray-500">{project.description}</p>

<div className="mt-4">

<span className="inline-flex items-center px-2.5 py-0.5 rounded-full text-xs font-medium bg-blue-100 text-blue-800">

{project.status}

</span>

</div>

</Link>

<div className="flex space-x-2 ml-4">

<button

onClick={(e) => handleEditClick(e, project)}

className="text-blue-600 hover:text-blue-800"

>

Edit

</button>

<button

onClick={(e) => handleDeleteClick(e, project)}

className="text-red-600 hover:text-red-800"

>

Delete

</button>

</div>

</div>

</div>

</div>

))}

</div>

)}

</main>

</div>

{/\* Edit Project Modal \*/}

<Modal

isOpen={editModalOpen}

onRequestClose={() => setEditModalOpen(false)}

className="bg-white rounded-lg p-6 max-w-2xl w-full mx-auto mt-20 focus:outline-none"

overlayClassName="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center"

style={{

content: {

maxHeight: '80vh',

overflowY: 'auto',

padding: '0',

position: 'relative',

inset: 'unset',

margin: 'auto',

display: 'flex',

flexDirection: 'column',

},

overlay: {

display: 'flex',

alignItems: 'center',

justifyContent: 'center',

zIndex: 1000,

}

}}

>

<div className="space-y-4 p-6">

<h3 className="text-xl font-bold text-gray-900">Edit Project</h3>

<form onSubmit={handleEditSubmit} className="space-y-4">

<div>

<label className="block text-sm font-medium text-gray-700">Project Name</label>

<input

type="text"

value={editFields.name}

onChange={(e) => setEditFields({ ...editFields, name: e.target.value })}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

required

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Description</label>

<textarea

value={editFields.description}

onChange={(e) => setEditFields({ ...editFields, description: e.target.value })}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

rows="4"

required

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Status</label>

<select

value={editFields.status}

onChange={(e) => setEditFields({ ...editFields, status: e.target.value })}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

>

<option value="Active">Active</option>

<option value="On Hold">On Hold</option>

<option value="Completed">Completed</option>

<option value="Cancelled">Cancelled</option>

</select>

</div>

<div className="mt-6">

<h4 className="text-lg font-medium text-gray-900 mb-4">Team Members</h4>

<div className="space-y-4">

{editTeamMembers.map((member, index) => (

<div key={index} className="flex items-center justify-between bg-gray-50 p-3 rounded-md">

<div>

<p className="font-medium">{member.name}</p>

<p className="text-sm text-gray-500">{member.email}</p>

</div>

<button

type="button"

onClick={() => handleRemoveTeamMember(member.email)}

className="text-red-600 hover:text-red-800"

>

Remove

</button>

</div>

))}

</div>

</div>

<div className="flex justify-end space-x-3 mt-6">

<button

type="button"

onClick={() => setEditModalOpen(false)}

className="px-4 py-2 bg-gray-200 text-gray-700 rounded-md hover:bg-gray-300"

>

Cancel

</button>

<button

type="submit"

className="px-4 py-2 bg-blue-600 text-white rounded-md hover:bg-blue-700"

>

Save Changes

</button>

</div>

</form>

<div className="mt-4 p-4 border border-gray-200 rounded-md">

<h5 className="font-medium mb-2">Add New Team Member</h5>

<div className="space-y-3">

<div>

<input

type="text"

placeholder="Name"

value={newTeamMember.name}

onChange={(e) => setNewTeamMember({ ...newTeamMember, name: e.target.value })}

className="block w-full border border-gray-300 rounded-md px-3 py-2"

required

/>

</div>

<div>

<input

type="email"

placeholder="Email"

value={newTeamMember.email}

onChange={(e) => setNewTeamMember({ ...newTeamMember, email: e.target.value })}

className="block w-full border border-gray-300 rounded-md px-3 py-2"

required

/>

</div>

<button

type="button"

onClick={handleAddTeamMember}

className="w-full bg-blue-600 text-white px-4 py-2 rounded-md hover:bg-blue-700"

>

Add Member

</button>

</div>

</div>

</div>

</Modal>

{/\* Delete Confirmation Modal \*/}

<Modal

isOpen={deleteModalOpen}

onRequestClose={() => setDeleteModalOpen(false)}

className="bg-white rounded-lg p-6 max-w-md mx-auto mt-20"

overlayClassName="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center"

>

<div className="space-y-4">

<h3 className="text-xl font-bold text-gray-900">Delete Project</h3>

<p className="text-gray-600">

Are you sure you want to delete "{selectedProjectForEdit?.name}"? This action cannot be undone.

</p>

<div className="flex justify-end space-x-3 mt-6">

<button

onClick={() => setDeleteModalOpen(false)}

className="px-4 py-2 bg-gray-200 text-gray-700 rounded-md hover:bg-gray-300"

>

Cancel

</button>

<button

onClick={handleDeleteConfirm}

className="px-4 py-2 bg-red-600 text-white rounded-md hover:bg-red-700"

>

Delete

</button>

</div>

</div>

</Modal>

</div>

);

};

export default Dashboard;

// src/pages/Login.js

import React, { useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useAuth } from '../contexts/AuthContext';

const Login = () => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [error, setError] = useState('');

const [loading, setLoading] = useState(false);

const navigate = useNavigate();

const { login } = useAuth();

const handleSubmit = async (e) => {

e.preventDefault();

try {

setError('');

setLoading(true);

await login(email, password);

setTimeout(() => navigate('/'), 100);

} catch (error) {

setError('Failed to sign in. Please check your credentials.');

} finally {

setLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-100 flex flex-col justify-center py-12 sm:px-6 lg:px-8">

<div className="sm:mx-auto sm:w-full sm:max-w-md">

<h2 className="mt-6 text-center text-3xl font-extrabold text-gray-900">

Sign in to Bug Tracker

</h2>

<p className="mt-2 text-center text-sm text-gray-600">

Or{' '}

<Link to="/register" className="font-medium text-blue-600 hover:text-blue-500">

create a new account

</Link>

</p>

</div>

<div className="mt-8 sm:mx-auto sm:w-full sm:max-w-md">

<div className="bg-white py-8 px-4 shadow sm:rounded-lg sm:px-10">

{error && (

<div className="mb-4 bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

)}

<form className="space-y-6" onSubmit={handleSubmit}>

<div>

<label htmlFor="email" className="block text-sm font-medium text-gray-700">

Email address

</label>

<div className="mt-1">

<input

id="email"

name="email"

type="email"

autoComplete="email"

required

value={email}

onChange={(e) => setEmail(e.target.value)}

className="appearance-none block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm"

placeholder="Enter your email"

/>

</div>

</div>

<div>

<label htmlFor="password" className="block text-sm font-medium text-gray-700">

Password

</label>

<div className="mt-1">

<input

id="password"

name="password"

type="password"

autoComplete="current-password"

required

value={password}

onChange={(e) => setPassword(e.target.value)}

className="appearance-none block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm"

placeholder="Enter your password"

/>

</div>

</div>

<div>

<button

type="submit"

disabled={loading}

className="w-full flex justify-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50"

>

{loading ? 'Signing in...' : 'Sign in'}

</button>

</div>

</form>

</div>

</div>

</div>

);

};

export default Login;

// src/pages/ProjectDetail.js

import React, { useState, useEffect } from 'react';

import { useParams, useNavigate } from 'react-router-dom';

import axios from 'axios';

import KanbanBoard from '../components/KanbanBoard';

const ProjectDetail = () => {

const [project, setProject] = useState(null);

const [loading, setLoading] = useState(true);

const [error, setError] = useState('');

const { id } = useParams();

const navigate = useNavigate();

useEffect(() => {

fetchProject();

}, [id]);

const fetchProject = async () => {

try {

const response = await axios.get(`http://localhost:5000/api/projects/${id}`, {

headers: {

Authorization: `Bearer ${localStorage.getItem('token')}`

}

});

setProject(response.data);

setLoading(false);

} catch (error) {

setError('Failed to fetch project details');

setLoading(false);

}

};

if (loading) {

return (

<div className="min-h-screen bg-gray-100 p-6">

<div className="max-w-7xl mx-auto">

<div className="text-center">Loading...</div>

</div>

</div>

);

}

if (error) {

return (

<div className="min-h-screen bg-gray-100 p-6">

<div className="max-w-7xl mx-auto">

<div className="bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

</div>

</div>

);

}

return (

<div className="min-h-screen bg-gray-100">

<div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-8">

<div className="mb-8">

<h1 className="text-3xl font-bold text-gray-900">{project.name}</h1>

<p className="mt-2 text-gray-600">{project.description}</p>

</div>

<div className="bg-white shadow rounded-lg">

<div className="px-4 py-5 sm:p-6">

<div className="flex justify-between items-center mb-6">

<h2 className="text-xl font-semibold text-gray-900">Tickets</h2>

<button

onClick={() => navigate(`/projects/${id}/tickets/new`)}

className="bg-blue-600 text-white px-4 py-2 rounded-md text-sm font-medium hover:bg-blue-700"

>

Create Ticket

</button>

</div>

<KanbanBoard projectId={id} />

</div>

</div>

</div>

</div>

);

};

export default ProjectDetail;

// src/pages/ProjectDetails.js

import React, { useState, useEffect } from 'react';

import { useParams, Link, useNavigate } from 'react-router-dom';

import axios from 'axios';

import { DragDropContext, Droppable, Draggable } from '@hello-pangea/dnd';

import Modal from 'react-modal';

const ProjectDetails = () => {

const { id } = useParams();

const navigate = useNavigate();

const [project, setProject] = useState(null);

const [tickets, setTickets] = useState([]);

const [loading, setLoading] = useState(true);

const [error, setError] = useState('');

const [statusFilter, setStatusFilter] = useState('');

const [priorityFilter, setPriorityFilter] = useState('');

const [assigneeFilter, setAssigneeFilter] = useState('');

const [search, setSearch] = useState('');

const [assignees, setAssignees] = useState([]);

const [editModalOpen, setEditModalOpen] = useState(false);

const [editTicket, setEditTicket] = useState(null);

const [editFields, setEditFields] = useState({ title: '', description: '', priority: '', status: '', assignee: '' });

const [deleteConfirmOpen, setDeleteConfirmOpen] = useState(false);

const [deleteTicketId, setDeleteTicketId] = useState(null);

const [currentUser, setCurrentUser] = useState({ role: 'user', email: '' });

const [editProjectModalOpen, setEditProjectModalOpen] = useState(false);

const [editProjectFields, setEditProjectFields] = useState({ name: '', description: '', status: '' });

const [newTeamMember, setNewTeamMember] = useState({ name: '', email: '' });

useEffect(() => {

fetchProject();

fetchTickets();

}, [id]);

useEffect(() => {

// Extract unique assignees for filter dropdown

setAssignees([...new Set(tickets.map(t => t.assignee?.name).filter(Boolean))]);

}, [tickets]);

useEffect(() => {

// Simulate fetching current user (replace with real auth)

setCurrentUser({ role: 'admin', email: 'admin@example.com' });

}, []);

const fetchProject = async () => {

try {

const response = await axios.get(`http://localhost:5000/api/projects/${id}`);

setProject(response.data);

} catch (error) {

setError('Failed to fetch project details');

}

};

const fetchTickets = async () => {

try {

let url = `http://localhost:5000/api/tickets/project/${id}?`;

if (statusFilter) url += `status=${statusFilter}&`;

if (priorityFilter) url += `priority=${priorityFilter}&`;

if (assigneeFilter) url += `assignee=${encodeURIComponent(assigneeFilter)}&`;

if (search) url += `q=${encodeURIComponent(search)}&`;

const response = await axios.get(url);

setTickets(response.data);

} catch (error) {

setError('Failed to fetch tickets');

} finally {

setLoading(false);

}

};

const onDragEnd = async (result) => {

if (!result.destination) return;

const { source, destination, draggableId } = result;

if (

source.droppableId === destination.droppableId &&

source.index === destination.index

) {

return;

}

const ticket = tickets.find((t) => t.\_id === draggableId);

const newStatus = destination.droppableId;

try {

await axios.put(`http://localhost:5000/api/tickets/${draggableId}`, {

status: newStatus,

});

const newTickets = tickets.map((t) =>

t.\_id === draggableId ? { ...t, status: newStatus } : t

);

setTickets(newTickets);

} catch (error) {

setError('Failed to update ticket status');

}

};

const openEditModal = (ticket) => {

setEditTicket(ticket);

setEditFields({

title: ticket.title,

description: ticket.description,

priority: ticket.priority,

status: ticket.status,

assignee: ticket.assignee?.name || ''

});

setEditModalOpen(true);

};

const closeEditModal = () => setEditModalOpen(false);

const handleEditChange = (field, value) => {

setEditFields({ ...editFields, [field]: value });

};

const handleEditSubmit = async (e) => {

e.preventDefault();

try {

await axios.put(`http://localhost:5000/api/tickets/${editTicket.\_id}`, {

title: editFields.title,

description: editFields.description,

priority: editFields.priority,

status: editFields.status,

assignee: tickets.find(t => t.assignee?.name === editFields.assignee)?.assignee || editTicket.assignee

});

setEditModalOpen(false);

fetchTickets();

} catch (error) {

setError('Failed to update ticket');

}

};

const openDeleteConfirm = (ticketId) => {

setDeleteTicketId(ticketId);

setDeleteConfirmOpen(true);

};

const closeDeleteConfirm = () => setDeleteConfirmOpen(false);

const handleDelete = async () => {

try {

await axios.delete(`http://localhost:5000/api/tickets/${deleteTicketId}`);

setDeleteConfirmOpen(false);

fetchTickets();

} catch (error) {

setError('Failed to delete ticket');

}

};

const openEditProjectModal = () => {

setEditProjectFields({

name: project.name,

description: project.description,

status: project.status

});

setEditProjectModalOpen(true);

};

const closeEditProjectModal = () => setEditProjectModalOpen(false);

const handleProjectEditChange = (field, value) => {

setEditProjectFields({ ...editProjectFields, [field]: value });

};

const handleProjectEditSubmit = async (e) => {

e.preventDefault();

try {

await axios.put(`http://localhost:5000/api/projects/${id}`, editProjectFields);

setEditProjectModalOpen(false);

fetchProject();

} catch (error) {

setError('Failed to update project');

}

};

const handleRemoveTeamMember = async (email) => {

try {

await axios.delete(`http://localhost:5000/api/projects/${id}/members/${email}`);

fetchProject();

} catch (error) {

setError('Failed to remove team member');

}

};

if (loading) {

return (

<div className="min-h-screen bg-gray-50 flex items-center justify-center">

<div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-primary-600"></div>

</div>

);

}

const columns = {

todo: tickets.filter((ticket) => ticket.status === 'todo'),

'in-progress': tickets.filter((ticket) => ticket.status === 'in-progress'),

done: tickets.filter((ticket) => ticket.status === 'done'),

};

return (

<div className="min-h-screen bg-gray-50">

<div className="max-w-7xl mx-auto py-6 sm:px-6 lg:px-8">

<div className="px-4 py-6 sm:px-0">

<div className="flex justify-between items-center mb-6">

<div>

<h2 className="text-3xl font-extrabold text-blue-700 mb-1">{project?.name}</h2>

<div className="h-1 w-16 bg-blue-400 rounded mb-2"></div>

<p className="mt-1 text-md text-gray-600">{project?.description}</p>

</div>

<div className="flex flex-col items-end space-y-2">

<button

onClick={openEditProjectModal}

className="mb-2 px-4 py-2 bg-blue-100 text-blue-700 rounded-md hover:bg-blue-200 focus:outline-none"

>

Edit Project

</button>

<button

onClick={() => navigate('/projects/new')}

className="mb-2 px-4 py-2 bg-blue-100 text-blue-700 rounded-md hover:bg-blue-200 focus:outline-none"

>

Back to Create Project

</button>

<Link

to={`/projects/${id}/tickets/new`}

className="inline-flex items-center px-4 py-2 border border-transparent text-sm font-medium rounded-md text-white bg-primary-600 hover:bg-primary-700"

>

Create Ticket

</Link>

</div>

</div>

{error && (

<div className="mb-4 bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

)}

<div className="flex flex-wrap gap-4 mb-6 items-center">

<input

type="text"

placeholder="Search tickets..."

value={search}

onChange={e => setSearch(e.target.value)}

onKeyDown={e => { if (e.key === 'Enter') fetchTickets(); }}

className="border border-gray-300 rounded px-3 py-2 text-sm focus:outline-none focus:ring-2 focus:ring-blue-500"

/>

<select

value={statusFilter}

onChange={e => { setStatusFilter(e.target.value); setLoading(true); setTimeout(fetchTickets, 0); }}

className="border border-gray-300 rounded px-3 py-2 text-sm"

>

<option value="">All Statuses</option>

<option value="todo">To Do</option>

<option value="in-progress">In Progress</option>

<option value="done">Done</option>

</select>

<select

value={priorityFilter}

onChange={e => { setPriorityFilter(e.target.value); setLoading(true); setTimeout(fetchTickets, 0); }}

className="border border-gray-300 rounded px-3 py-2 text-sm"

>

<option value="">All Priorities</option>

<option value="low">Low</option>

<option value="medium">Medium</option>

<option value="high">High</option>

<option value="urgent">Urgent</option>

</select>

<select

value={assigneeFilter}

onChange={e => { setAssigneeFilter(e.target.value); setLoading(true); setTimeout(fetchTickets, 0); }}

className="border border-gray-300 rounded px-3 py-2 text-sm"

>

<option value="">All Assignees</option>

{assignees.map(name => (

<option key={name} value={name}>{name}</option>

))}

</select>

<button

onClick={fetchTickets}

className="bg-blue-600 text-white px-4 py-2 rounded-md text-sm font-medium hover:bg-blue-700"

>

Filter

</button>

<button

onClick={() => {

setStatusFilter('');

setPriorityFilter('');

setAssigneeFilter('');

setSearch('');

setLoading(true);

setTimeout(fetchTickets, 0);

}}

className="bg-gray-300 text-gray-700 px-4 py-2 rounded-md text-sm font-medium hover:bg-gray-400"

>

Reset

</button>

</div>

{tickets.length === 0 && (

<div className="text-center text-gray-500 mt-8">No tickets found for the selected filters.</div>

)}

<DragDropContext onDragEnd={onDragEnd}>

<div className="grid grid-cols-1 gap-6 sm:grid-cols-3">

{Object.entries(columns).map(([status, tickets]) => (

<div key={status} className="bg-gray-100 rounded-lg p-4">

<h3 className="text-lg font-medium text-gray-900 mb-4 capitalize">

{status.replace('-', ' ')}

</h3>

<Droppable droppableId={status}>

{(provided) => (

<div

{...provided.droppableProps}

ref={provided.innerRef}

className="space-y-4"

>

{tickets.map((ticket, index) => (

<Draggable

key={ticket.\_id}

draggableId={ticket.\_id}

index={index}

>

{(provided) => (

<div

ref={provided.innerRef}

{...provided.draggableProps}

{...provided.dragHandleProps}

className="bg-white p-4 rounded-lg shadow"

>

<Link

to={`/tickets/${ticket.\_id}`}

className="block hover:bg-gray-50"

>

<h4 className="text-sm font-medium text-gray-900">

{ticket.title}

</h4>

<div className="mt-2 flex items-center justify-between">

<span

className={`inline-flex items-center px-2.5 py-0.5 rounded-full text-xs font-medium ${ticket.priority === 'high'

? 'bg-red-100 text-red-800'

: ticket.priority === 'medium'

? 'bg-yellow-100 text-yellow-800'

: 'bg-green-100 text-green-800'

}`}

>

{ticket.priority}

</span>

{ticket.assignee && (

<span className="text-xs text-gray-500">

{ticket.assignee.name}

</span>

)}

</div>

</Link>

{(currentUser.role === 'admin' || currentUser.email === ticket.assignee?.email) && (

<div className="flex gap-2 mt-2">

<button

onClick={() => openEditModal(ticket)}

className="text-blue-600 text-xs hover:underline"

>

Edit

</button>

<button

onClick={() => openDeleteConfirm(ticket.\_id)}

className="text-red-600 text-xs hover:underline"

>

Delete

</button>

</div>

)}

</div>

)}

</Draggable>

))}

{provided.placeholder}

</div>

)}

</Droppable>

</div>

))}

</div>

</DragDropContext>

</div>

</div>

<Modal

isOpen={editModalOpen}

onRequestClose={closeEditModal}

contentLabel="Edit Ticket"

className="fixed inset-0 flex items-center justify-center z-50"

overlayClassName="fixed inset-0 bg-black bg-opacity-30 z-40"

ariaHideApp={false}

>

<div className="bg-white rounded-lg shadow-lg p-8 w-full max-w-lg">

<h2 className="text-xl font-bold mb-4">Edit Ticket</h2>

<form onSubmit={handleEditSubmit} className="space-y-4">

<div>

<label className="block text-sm font-medium text-gray-700">Title</label>

<input

type="text"

value={editFields.title}

onChange={e => handleEditChange('title', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

required

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Description</label>

<textarea

value={editFields.description}

onChange={e => handleEditChange('description', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

rows={3}

required

/>

</div>

<div className="grid grid-cols-1 gap-4 sm:grid-cols-2">

<div>

<label className="block text-sm font-medium text-gray-700">Priority</label>

<select

value={editFields.priority}

onChange={e => handleEditChange('priority', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

>

<option value="low">Low</option>

<option value="medium">Medium</option>

<option value="high">High</option>

<option value="urgent">Urgent</option>

</select>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Status</label>

<select

value={editFields.status}

onChange={e => handleEditChange('status', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

>

<option value="todo">To Do</option>

<option value="in-progress">In Progress</option>

<option value="done">Done</option>

</select>

</div>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Assignee</label>

<select

value={editFields.assignee}

onChange={e => handleEditChange('assignee', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

required

>

<option value="">Select team member</option>

{assignees.map(name => (

<option key={name} value={name}>{name}</option>

))}

</select>

</div>

<div className="flex justify-end space-x-2">

<button

type="button"

onClick={closeEditModal}

className="px-4 py-2 bg-gray-200 text-gray-700 rounded-md hover:bg-gray-300"

>

Cancel

</button>

<button

type="submit"

className="px-4 py-2 bg-blue-600 text-white rounded-md hover:bg-blue-700"

>

Save Changes

</button>

</div>

</form>

</div>

</Modal>

<Modal

isOpen={deleteConfirmOpen}

onRequestClose={closeDeleteConfirm}

contentLabel="Delete Ticket"

className="fixed inset-0 flex items-center justify-center z-50"

overlayClassName="fixed inset-0 bg-black bg-opacity-30 z-40"

ariaHideApp={false}

>

<div className="bg-white rounded-lg shadow-lg p-8 w-full max-w-md">

<h2 className="text-xl font-bold mb-4 text-red-600">Delete Ticket</h2>

<p>Are you sure you want to delete this ticket? This action cannot be undone.</p>

<div className="flex justify-end space-x-2 mt-6">

<button

type="button"

onClick={closeDeleteConfirm}

className="px-4 py-2 bg-gray-200 text-gray-700 rounded-md hover:bg-gray-300"

>

Cancel

</button>

<button

type="button"

onClick={handleDelete}

className="px-4 py-2 bg-red-600 text-white rounded-md hover:bg-red-700"

>

Delete

</button>

</div>

</div>

</Modal>

<Modal

isOpen={editProjectModalOpen}

onRequestClose={closeEditProjectModal}

className="bg-white rounded-lg p-6 max-w-2xl w-full mx-auto mt-20 focus:outline-none"

overlayClassName="fixed inset-0 bg-black bg-opacity-50 flex items-center justify-center"

style={{

content: {

maxHeight: '80vh',

overflowY: 'auto',

padding: '0',

position: 'relative',

inset: 'unset',

margin: 'auto',

display: 'flex',

flexDirection: 'column',

},

overlay: {

display: 'flex',

alignItems: 'center',

justifyContent: 'center',

zIndex: 1000,

}

}}

>

<div className="space-y-4 p-6">

<h3 className="text-xl font-bold text-gray-900">Edit Project</h3>

<form onSubmit={handleProjectEditSubmit} className="space-y-4">

<div>

<label className="block text-sm font-medium text-gray-700">Project Name</label>

<input

type="text"

value={editProjectFields.name}

onChange={(e) => handleProjectEditChange('name', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

required

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Description</label>

<textarea

value={editProjectFields.description}

onChange={(e) => handleProjectEditChange('description', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

rows="4"

required

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Status</label>

<select

value={editProjectFields.status}

onChange={(e) => handleProjectEditChange('status', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

>

<option value="Active">Active</option>

<option value="On Hold">On Hold</option>

<option value="Completed">Completed</option>

<option value="Cancelled">Cancelled</option>

</select>

</div>

<div className="mt-6">

<h4 className="text-lg font-medium text-gray-900 mb-4">Team Members</h4>

<div className="space-y-4">

{project?.teamMembers.map((member, index) => (

<div key={index} className="flex items-center justify-between bg-gray-50 p-3 rounded-md">

<div>

<p className="font-medium">{member.name}</p>

<p className="text-sm text-gray-500">{member.email}</p>

</div>

<button

type="button"

onClick={() => handleRemoveTeamMember(member.email)}

className="text-red-600 hover:text-red-800"

>

Remove

</button>

</div>

))}

</div>

</div>

<div className="flex justify-end space-x-3 mt-6">

<button

type="button"

onClick={closeEditProjectModal}

className="px-4 py-2 bg-gray-200 text-gray-700 rounded-md hover:bg-gray-300"

>

Cancel

</button>

<button

type="submit"

className="px-4 py-2 bg-blue-600 text-white rounded-md hover:bg-blue-700"

>

Save Changes

</button>

</div>

</form>

</div>

</Modal>

</div>

);

};

export default ProjectDetails;

// src/pages/Register.js

import React, { useState } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useAuth } from '../contexts/AuthContext';

const Register = () => {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [confirmPassword, setConfirmPassword] = useState('');

const [name, setName] = useState('');

const [error, setError] = useState('');

const [loading, setLoading] = useState(false);

const navigate = useNavigate();

const { register } = useAuth();

const handleSubmit = async (e) => {

e.preventDefault();

if (password !== confirmPassword) {

return setError('Passwords do not match');

}

try {

setError('');

setLoading(true);

await register(email, password, name);

navigate('/');

} catch (error) {

setError('Failed to create an account. Please try again.');

} finally {

setLoading(false);

}

};

return (

<div className="min-h-screen bg-gray-100 flex flex-col justify-center py-12 sm:px-6 lg:px-8">

<div className="sm:mx-auto sm:w-full sm:max-w-md">

<h2 className="mt-6 text-center text-3xl font-extrabold text-gray-900">

Create your account

</h2>

<p className="mt-2 text-center text-sm text-gray-600">

Or{' '}

<Link to="/login" className="font-medium text-blue-600 hover:text-blue-500">

sign in to your account

</Link>

</p>

</div>

<div className="mt-8 sm:mx-auto sm:w-full sm:max-w-md">

<div className="bg-white py-8 px-4 shadow sm:rounded-lg sm:px-10">

{error && (

<div className="mb-4 bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

)}

<form className="space-y-6" onSubmit={handleSubmit}>

<div>

<label htmlFor="name" className="block text-sm font-medium text-gray-700">

Full Name

</label>

<div className="mt-1">

<input

id="name"

name="name"

type="text"

required

value={name}

onChange={(e) => setName(e.target.value)}

className="appearance-none block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm"

placeholder="Enter your full name"

/>

</div>

</div>

<div>

<label htmlFor="email" className="block text-sm font-medium text-gray-700">

Email address

</label>

<div className="mt-1">

<input

id="email"

name="email"

type="email"

autoComplete="email"

required

value={email}

onChange={(e) => setEmail(e.target.value)}

className="appearance-none block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm"

placeholder="Enter your email"

/>

</div>

</div>

<div>

<label htmlFor="password" className="block text-sm font-medium text-gray-700">

Password

</label>

<div className="mt-1">

<input

id="password"

name="password"

type="password"

required

value={password}

onChange={(e) => setPassword(e.target.value)}

className="appearance-none block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm"

placeholder="Create a password"

/>

</div>

</div>

<div>

<label htmlFor="confirmPassword" className="block text-sm font-medium text-gray-700">

Confirm Password

</label>

<div className="mt-1">

<input

id="confirmPassword"

name="confirmPassword"

type="password"

required

value={confirmPassword}

onChange={(e) => setConfirmPassword(e.target.value)}

className="appearance-none block w-full px-3 py-2 border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-blue-500 focus:border-blue-500 sm:text-sm"

placeholder="Confirm your password"

/>

</div>

</div>

<div>

<button

type="submit"

disabled={loading}

className="w-full flex justify-center py-2 px-4 border border-transparent rounded-md shadow-sm text-sm font-medium text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500 disabled:opacity-50"

>

{loading ? 'Creating account...' : 'Create account'}

</button>

</div>

</form>

</div>

</div>

</div>

);

};

export default Register;

// src/pages/TicketDetails.js

import React, { useState, useEffect } from 'react';

import { useParams, useNavigate } from 'react-router-dom';

import axios from 'axios';

// import { useAuth } from '../contexts/AuthContext';

import Modal from 'react-modal';

function buildThread(comments) {

const map = {};

comments.forEach(c => map[c.\_id] = { ...c, replies: [] });

const roots = [];

comments.forEach(c => {

if (c.parentId) {

map[c.parentId]?.replies.push(map[c.\_id]);

} else {

roots.push(map[c.\_id]);

}

});

return roots;

}

function CommentThread({ comments, onReply }) {

return comments.map(comment => (

<div key={comment.\_id} style={{ marginLeft: comment.parentId ? 20 : 0 }} className="mb-2">

<div className="text-xs text-gray-500">

<b>{comment.userId}</b> {/\* Replace with user name if you populate \*/}

<span> {new Date(comment.createdAt).toLocaleString()}</span>

</div>

<div className="mb-1">{comment.text}</div>

<button className="text-blue-600 text-xs mb-1" onClick={() => onReply(comment.\_id)}>Reply</button>

{comment.replies && comment.replies.length > 0 && (

<CommentThread comments={comment.replies} onReply={onReply} />

)}

</div>

));

}

const TicketDetails = () => {

const { id } = useParams();

const [ticket, setTicket] = useState(null);

const [comment, setComment] = useState('');

const [loading, setLoading] = useState(true);

const [error, setError] = useState('');

const [submitting, setSubmitting] = useState(false);

const navigate = useNavigate();

// const { user } = useAuth();

const [editModalOpen, setEditModalOpen] = useState(false);

const [editFields, setEditFields] = useState({ title: '', description: '', priority: '', status: '', assignee: '' });

const [teamMembers, setTeamMembers] = useState([]);

const [comments, setComments] = useState([]);

const [replyTo, setReplyTo] = useState(null);

useEffect(() => {

fetchTicket();

fetchComments();

const interval = setInterval(fetchComments, 3000); // Poll every 3s

return () => clearInterval(interval);

// eslint-disable-next-line react-hooks/exhaustive-deps

}, [id]);

useEffect(() => {

if (ticket?.project) {

axios.get(`http://localhost:5000/api/projects/${ticket.project}`)

.then(res => setTeamMembers(res.data.teamMembers || []));

}

}, [ticket?.project]);

const fetchTicket = async () => {

try {

const response = await axios.get(`http://localhost:5000/api/tickets/${id}`);

setTicket(response.data);

} catch (error) {

setError('Failed to fetch ticket details');

} finally {

setLoading(false);

}

};

const fetchComments = async () => {

try {

const response = await axios.get(`http://localhost:5000/api/comments/ticket/${id}`);

setComments(response.data);

} catch (error) {

// Optionally handle error

}

};

const handleStatusChange = async (newStatus) => {

try {

await axios.put(`http://localhost:5000/api/tickets/${id}`, {

status: newStatus,

});

setTicket({ ...ticket, status: newStatus });

} catch (error) {

setError('Failed to update ticket status');

}

};

const handlePriorityChange = async (newPriority) => {

try {

await axios.put(`http://localhost:5000/api/tickets/${id}`, {

priority: newPriority,

});

setTicket({ ...ticket, priority: newPriority });

} catch (error) {

setError('Failed to update ticket priority');

}

};

const handleCommentSubmit = async (e) => {

e.preventDefault();

if (!comment.trim()) return;

try {

setSubmitting(true);

await axios.post(`http://localhost:5000/api/comments`, {

ticketId: id,

text: comment,

parentId: replyTo

});

setComment('');

setReplyTo(null);

fetchComments(); // Refresh comments

} catch (error) {

setError('Failed to add comment');

console.error(error?.response?.data || error);

} finally {

setSubmitting(false);

}

};

const openEditModal = () => {

setEditFields({

title: ticket.title,

description: ticket.description,

priority: ticket.priority,

status: ticket.status,

assignee: ticket.assignee?.email || ''

});

setEditModalOpen(true);

};

const closeEditModal = () => setEditModalOpen(false);

const handleEditChange = (field, value) => {

setEditFields({ ...editFields, [field]: value });

};

const handleEditSubmit = async (e) => {

e.preventDefault();

const selectedAssignee = teamMembers.find(tm => tm.email === editFields.assignee);

try {

await axios.put(`http://localhost:5000/api/tickets/${id}`, {

title: editFields.title,

description: editFields.description,

priority: editFields.priority,

status: editFields.status,

assignee: selectedAssignee

});

setTicket({ ...ticket, ...editFields, assignee: selectedAssignee });

setEditModalOpen(false);

} catch (error) {

setError('Failed to update ticket');

}

};

if (loading) {

return (

<div className="min-h-screen bg-gray-50 flex items-center justify-center">

<div className="animate-spin rounded-full h-12 w-12 border-t-2 border-b-2 border-primary-600"></div>

</div>

);

}

return (

<div className="min-h-screen bg-gray-50 py-12 px-4 sm:px-6 lg:px-8">

<div className="max-w-3xl mx-auto">

<div className="md:flex md:items-center md:justify-between">

<div className="flex-1 min-w-0">

<h2 className="text-2xl font-bold leading-7 text-gray-900 sm:text-3xl sm:truncate">

{ticket?.title}

</h2>

</div>

<div className="mt-4 flex md:mt-0 md:ml-4 space-x-2">

<button

onClick={() => navigate(`/projects/${ticket?.project}`)}

className="inline-flex items-center px-4 py-2 border border-gray-300 shadow-sm text-sm font-medium rounded-md text-gray-700 bg-white hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary-500"

>

Back to Project

</button>

<button

onClick={openEditModal}

className="inline-flex items-center px-4 py-2 border border-blue-500 shadow-sm text-sm font-medium rounded-md text-blue-700 bg-blue-100 hover:bg-blue-200 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-blue-500"

>

Edit Ticket

</button>

</div>

</div>

{error && (

<div className="mt-4 bg-red-50 border border-red-200 text-red-600 px-4 py-3 rounded">

{error}

</div>

)}

<div className="mt-8 bg-white shadow sm:rounded-lg">

<div className="px-4 py-5 sm:p-6">

<div className="grid grid-cols-1 gap-6 sm:grid-cols-2">

<div>

<label className="block text-sm font-medium text-gray-700">Status</label>

<select

value={ticket?.status}

onChange={(e) => handleStatusChange(e.target.value)}

className="mt-1 block w-full pl-3 pr-10 py-2 text-base border-gray-300 focus:outline-none focus:ring-primary-500 focus:border-primary-500 sm:text-sm rounded-md"

>

<option value="todo">To Do</option>

<option value="in-progress">In Progress</option>

<option value="done">Done</option>

</select>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Priority</label>

<select

value={ticket?.priority}

onChange={(e) => handlePriorityChange(e.target.value)}

className="mt-1 block w-full pl-3 pr-10 py-2 text-base border-gray-300 focus:outline-none focus:ring-primary-500 focus:border-primary-500 sm:text-sm rounded-md"

>

<option value="low">Low</option>

<option value="medium">Medium</option>

<option value="high">High</option>

<option value="urgent">Urgent</option>

</select>

</div>

</div>

<div className="mt-6">

<h3 className="text-lg font-medium text-gray-900">Description</h3>

<div className="mt-2 text-sm text-gray-500 whitespace-pre-wrap">

{ticket?.description}

</div>

</div>

<div className="mt-6">

<h3 className="text-lg font-medium text-gray-900">Comments</h3>

<div className="mt-4 space-y-4">

<CommentThread comments={buildThread(comments)} onReply={setReplyTo} />

</div>

<form onSubmit={handleCommentSubmit} className="mt-4 flex flex-col gap-2">

{replyTo && (

<div className="text-xs text-blue-600">Replying to comment {replyTo} <button type="button" onClick={() => setReplyTo(null)} className="ml-2 text-red-500">Cancel</button></div>

)}

<div>

<label htmlFor="comment" className="sr-only">

Add a comment

</label>

<textarea

id="comment"

name="comment"

rows={3}

value={comment}

onChange={(e) => setComment(e.target.value)}

className="shadow-sm focus:ring-primary-500 focus:border-primary-500 block w-full sm:text-sm border-gray-300 rounded-md"

placeholder="Add a comment..."

/>

</div>

<div className="flex justify-end">

<button

type="submit"

disabled={submitting || !comment.trim()}

className="inline-flex items-center px-4 py-2 border border-transparent text-sm font-medium rounded-md text-white bg-primary-600 hover:bg-primary-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-primary-500 disabled:opacity-50"

>

{submitting ? 'Posting...' : 'Add Comment'}

</button>

</div>

</form>

</div>

</div>

</div>

</div>

<Modal

isOpen={editModalOpen}

onRequestClose={closeEditModal}

contentLabel="Edit Ticket"

className="fixed inset-0 flex items-center justify-center z-50"

overlayClassName="fixed inset-0 bg-black bg-opacity-30 z-40"

ariaHideApp={false}

>

<div className="bg-white rounded-lg shadow-lg p-8 w-full max-w-lg">

<h2 className="text-xl font-bold mb-4">Edit Ticket</h2>

<form onSubmit={handleEditSubmit} className="space-y-4">

<div>

<label className="block text-sm font-medium text-gray-700">Title</label>

<input

type="text"

value={editFields.title}

onChange={e => handleEditChange('title', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

required

/>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Description</label>

<textarea

value={editFields.description}

onChange={e => handleEditChange('description', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

rows={3}

required

/>

</div>

<div className="grid grid-cols-1 gap-4 sm:grid-cols-2">

<div>

<label className="block text-sm font-medium text-gray-700">Priority</label>

<select

value={editFields.priority}

onChange={e => handleEditChange('priority', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

>

<option value="low">Low</option>

<option value="medium">Medium</option>

<option value="high">High</option>

<option value="urgent">Urgent</option>

</select>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Status</label>

<select

value={editFields.status}

onChange={e => handleEditChange('status', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

>

<option value="todo">To Do</option>

<option value="in-progress">In Progress</option>

<option value="done">Done</option>

</select>

</div>

</div>

<div>

<label className="block text-sm font-medium text-gray-700">Assignee</label>

<select

value={editFields.assignee}

onChange={e => handleEditChange('assignee', e.target.value)}

className="mt-1 block w-full border border-gray-300 rounded-md px-3 py-2"

required

>

<option value="">Select team member</option>

{teamMembers.map(tm => (

<option key={tm.email} value={tm.email}>

{tm.name} ({tm.email})

</option>

))}

</select>

</div>

<div className="flex justify-end space-x-2">

<button

type="button"

onClick={closeEditModal}

className="px-4 py-2 bg-gray-200 text-gray-700 rounded-md hover:bg-gray-300"

>

Cancel

</button>

<button

type="submit"

className="px-4 py-2 bg-blue-600 text-white rounded-md hover:bg-blue-700"

>

Save Changes

</button>

</div>

</form>

</div>

</Modal>

</div>

);

};

export default TicketDetails;

// src/App.js

import React from 'react';

import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';

import { AuthProvider } from './contexts/AuthContext';

import PrivateRoute from './components/PrivateRoute';

import Login from './pages/Login';

import Register from './pages/Register';

import Dashboard from './pages/Dashboard';

import ProjectDetails from './pages/ProjectDetails';

import CreateProject from './pages/CreateProject';

import CreateTicket from './pages/CreateTicket';

import TicketDetails from './pages/TicketDetails';

function App() {

return (

<AuthProvider>

<div className="min-h-screen bg-gray-100">

<div className="p-4 bg-blue-500 text-white shadow-md">

<h1 className="text-2xl font-bold">Bug Tracker</h1>

</div>

<Router>

<Routes>

<Route path="/login" element={<Login />} />

<Route path="/register" element={<Register />} />

<Route

path="/"

element={

<PrivateRoute>

<Dashboard />

</PrivateRoute>

}

/>

<Route

path="/projects/new"

element={

<PrivateRoute>

<CreateProject />

</PrivateRoute>

}

/>

<Route

path="/projects/:id"

element={

<PrivateRoute>

<ProjectDetails />

</PrivateRoute>

}

/>

<Route

path="/projects/:projectId/tickets/new"

element={

<PrivateRoute>

<CreateTicket />

</PrivateRoute>

}

/>

<Route

path="/tickets/:id"

element={

<PrivateRoute>

<TicketDetails />

</PrivateRoute>

}

/>

<Route path="\*" element={<Navigate to="/" replace />} />

</Routes>

</Router>

</div>

</AuthProvider>

);

}

export default App;

// src/index.js

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);