const mongoose = require('mongoose');

const rubricCriteriaSchema = new mongoose.Schema({

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

description: String,

maxScore: { type: Number, required: true }

});

const assignmentSchema = new mongoose.Schema({

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

description: String,

subject: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Subject',

required: true

},

class: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Class',

required: true

},

teacher: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Teacher',

required: true

},

school: {

type: mongoose.Schema.Types.ObjectId,

ref: 'School',

required: true

},

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

attachments: [String], // Array of file URLs

videoUrl: String,

rubric: [rubricCriteriaSchema], // Rubric-based marking

maxPoints: { type: Number, default: 100 },

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

peerReviewEnabled: { type: Boolean, default: false }

}, {

timestamps: true

});

const submissionSchema = new mongoose.Schema({

assignment: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Assignment',

required: true

},

student: {

type: mongoose.Schema.Types.ObjectId,

ref: 'Student',

required: true

},

fileUrl: String,

videoUrl: String, // Student video submission

remarks: String,

grade: Number,

feedback: String,

feedbackVideoUrl: String, // Teacher video feedback

rubricScores: [{

criteriaId: mongoose.Schema.Types.ObjectId,

score: Number

}],

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

extensionGranted: Date,

gradedAt: Date

}, {

timestamps: true

});

module.exports = {

Assignment: mongoose.model('Assignment', assignmentSchema),

AssignmentSubmission: mongoose.model('AssignmentSubmission', submissionSchema)

};

const { Assignment, AssignmentSubmission } = require('../models/assignment.model');

const multer = require('multer');

const path = require('path');

const fs = require('fs');

const ffmpeg = require('fluent-ffmpeg');

const ffmpegPath = require('ffmpeg-static');

const { promisify } = require('util');

const writeFileAsync = promisify(fs.writeFile);

const unlinkAsync = promisify(fs.unlink);

ffmpeg.setFfmpegPath(ffmpegPath);

// Configure multer for file uploads

const storage = multer.diskStorage({

destination: (req, file, cb) => {

const type = file.fieldname.includes('video') ? 'videos' : 'assignments';

const uploadDir = `uploads/${type}`;

if (!fs.existsSync(uploadDir)) {

fs.mkdirSync(uploadDir, { recursive: true });

}

cb(null, uploadDir);

},

filename: (req, file, cb) => {

const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() \* 1E9);

cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname));

}

});

const upload = multer({

storage: storage,

fileFilter: (req, file, cb) => {

const filetypes = /jpeg|jpg|png|gif|pdf|doc|docx|mp4|mov|avi|mkv|webm/;

const extname = filetypes.test(path.extname(file.originalname).toLowerCase());

const mimetype = filetypes.test(file.mimetype);

if (extname && mimetype) {

return cb(null, true);

} else {

cb('Error: Only documents, images, and videos are allowed!');

}

},

limits: { fileSize: 500 \* 1024 \* 1024 } // 500MB

}).fields([

{ name: 'attachments', maxCount: 5 },

{ name: 'submission', maxCount: 1 },

{ name: 'feedbackVideo', maxCount: 1 },

{ name: 'assignmentVideo', maxCount: 1 }

]);

// Helper to handle file uploads

const handleUpload = (req, res, next) => {

upload(req, res, (err) => {

if (err) {

return res.status(400).json({ success: false, message: err });

}

next();

});

};

// Compress video helper

const compressVideo = async (filePath) => {

const tempPath = filePath + '.compressed.mp4';

return new Promise((resolve, reject) => {

ffmpeg(filePath)

.outputOptions([

'-c:v libx264',

'-crf 28',

'-preset medium',

'-c:a aac',

'-b:a 64k'

])

.on('end', () => resolve(tempPath))

.on('error', (err) => reject(err))

.save(tempPath);

});

};

// TEACHER creates an assignment

exports.createAssignment = [handleUpload, async (req, res) => {

try {

const {

title,

description,

subject,

class: classId,

dueDate,

videoUrl,

rubric,

maxPoints,

allowLateSubmission,

peerReviewEnabled

} = req.body;

// Handle file uploads

const attachments = [];

if (req.files && req.files['attachments']) {

req.files['attachments'].forEach(file => {

attachments.push(`/uploads/assignments/${file.filename}`);

});

}

// Handle video upload

let assignmentVideoUrl = videoUrl;

if (req.files && req.files['assignmentVideo']) {

const file = req.files['assignmentVideo'][0];

assignmentVideoUrl = `/uploads/videos/${file.filename}`;

// Compress video in background

compressVideo(file.path)

.then(compressedPath => {

fs.rename(compressedPath, file.path, (err) => {

if (err) console.error('Error replacing video:', err);

});

})

.catch(err => console.error('Compression error:', err));

}

const assignment = new Assignment({

title,

description,

subject,

class: classId,

dueDate,

attachments,

videoUrl: assignmentVideoUrl,

rubric: rubric ? JSON.parse(rubric) : [],

maxPoints,

allowLateSubmission,

peerReviewEnabled,

school: req.user.schoolId,

teacher: req.user.id

});

await assignment.save();

res.status(201).json({

success: true,

message: "Assignment created",

data: assignment

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error creating assignment",

error: error.message

});

}

}];

// Get single assignment

exports.getAssignment = async (req, res) => {

try {

const assignment = await Assignment.findById(req.params.id)

.populate('subject', 'subject\_name')

.populate('class', 'class\_text')

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

if (!assignment) {

return res.status(404).json({

success: false,

message: "Assignment not found"

});

}

res.status(200).json({

success: true,

data: assignment

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error fetching assignment",

error: error.message

});

}

};

// Update assignment

exports.updateAssignment = [handleUpload, async (req, res) => {

try {

const {

title,

description,

subject,

class: classId,

dueDate,

videoUrl,

rubric,

maxPoints,

allowLateSubmission,

peerReviewEnabled

} = req.body;

// Handle file uploads

const attachments = [];

if (req.files && req.files['attachments']) {

req.files['attachments'].forEach(file => {

attachments.push(`/uploads/assignments/${file.filename}`);

});

}

// Handle video upload

let assignmentVideoUrl = videoUrl;

if (req.files && req.files['assignmentVideo']) {

const file = req.files['assignmentVideo'][0];

assignmentVideoUrl = `/uploads/videos/${file.filename}`;

// Compress video in background

compressVideo(file.path)

.then(compressedPath => {

fs.rename(compressedPath, file.path, (err) => {

if (err) console.error('Error replacing video:', err);

});

})

.catch(err => console.error('Compression error:', err));

}

const updateData = {

title,

description,

subject,

class: classId,

dueDate,

videoUrl: assignmentVideoUrl,

rubric: rubric ? JSON.parse(rubric) : [],

maxPoints,

allowLateSubmission,

peerReviewEnabled,

$push: { attachments: { $each: attachments } }

};

// Remove null fields

Object.keys(updateData).forEach(key => {

if (updateData[key] === null || updateData[key] === undefined) {

delete updateData[key];

}

});

const assignment = await Assignment.findByIdAndUpdate(

req.params.id,

updateData,

{ new: true }

);

if (!assignment) {

return res.status(404).json({

success: false,

message: "Assignment not found"

});

}

res.status(200).json({

success: true,

message: "Assignment updated",

data: assignment

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error updating assignment",

error: error.message

});

}

}];

// Delete assignment

exports.deleteAssignment = async (req, res) => {

try {

const assignment = await Assignment.findByIdAndDelete(req.params.id);

if (!assignment) {

return res.status(404).json({

success: false,

message: "Assignment not found"

});

}

// Delete associated files

const deletePromises = [];

if (assignment.attachments && assignment.attachments.length > 0) {

assignment.attachments.forEach(file => {

const filePath = path.join(\_\_dirname, '..', file);

if (fs.existsSync(filePath)) {

deletePromises.push(unlinkAsync(filePath));

}

});

}

if (assignment.videoUrl) {

const videoPath = path.join(\_\_dirname, '..', assignment.videoUrl);

if (fs.existsSync(videoPath)) {

deletePromises.push(unlinkAsync(videoPath));

}

}

await Promise.all(deletePromises);

// Delete associated submissions

await AssignmentSubmission.deleteMany({ assignment: req.params.id });

res.status(200).json({

success: true,

message: "Assignment deleted successfully"

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error deleting assignment",

error: error.message

});

}

};

// STUDENT fetches assignments for their class - Updated version

exports.getAssignmentsForStudent = async (req, res) => {

try {

// Get student's class from their profile

const student = await Student.findById(req.user.id).select('class');

if (!student || !student.class) {

return res.status(400).json({

success: false,

message: "Student class information not found"

});

}

const assignments = await Assignment.find({

class: student.class,

school: req.user.schoolId

})

.populate('subject', 'subject\_name')

.populate('teacher', 'name')

.populate('class', 'class\_text')

.sort({ dueDate: 1 });

res.status(200).json({

success: true,

data: assignments

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error fetching assignments",

error: error.message

});

}

};

// STUDENT submits an assignment

exports.submitAssignment = [handleUpload, async (req, res) => {

try {

const { assignmentId, remarks, videoUrl } = req.body;

let fileUrl = '';

let submissionVideoUrl = videoUrl;

if (req.files && req.files['submission']) {

fileUrl = `/uploads/assignments/${req.files['submission'][0].filename}`;

}

if (req.files && req.files['submissionVideo']) {

const file = req.files['submissionVideo'][0];

submissionVideoUrl = `/uploads/videos/${file.filename}`;

// Compress video in background

compressVideo(file.path)

.then(compressedPath => {

fs.rename(compressedPath, file.path, (err) => {

if (err) console.error('Error replacing video:', err);

});

})

.catch(err => console.error('Compression error:', err));

}

// Check if submission is late

const assignment = await Assignment.findById(assignmentId);

const isLate = assignment && new Date() > assignment.dueDate;

if (isLate && !assignment.allowLateSubmission) {

return res.status(400).json({

success: false,

message: "Late submissions not allowed for this assignment"

});

}

const submission = new AssignmentSubmission({

assignment: assignmentId,

student: req.user.id,

fileUrl,

videoUrl: submissionVideoUrl,

remarks,

lateSubmission: isLate

});

await submission.save();

res.status(201).json({

success: true,

message: "Assignment submitted",

data: submission

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error submitting assignment",

error: error.message

});

}

}];

// TEACHER gets all submissions for an assignment

exports.getSubmissionsForAssignment = async (req, res) => {

try {

const { assignmentId } = req.params;

const submissions = await AssignmentSubmission.find({ assignment: assignmentId })

.populate('student', 'name email image\_url')

.sort({ createdAt: -1 });

res.status(200).json({

success: true,

data: submissions

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error fetching submissions",

error: error.message

});

}

};

// TEACHER grades a submission

exports.gradeSubmission = [handleUpload, async (req, res) => {

try {

const { submissionId } = req.params;

const { grade, feedback, rubricScores } = req.body;

let feedbackVideoUrl = '';

if (req.files && req.files['feedbackVideo']) {

const file = req.files['feedbackVideo'][0];

feedbackVideoUrl = `/uploads/videos/${file.filename}`;

// Compress video in background

compressVideo(file.path)

.then(compressedPath => {

fs.rename(compressedPath, file.path, (err) => {

if (err) console.error('Error replacing video:', err);

});

})

.catch(err => console.error('Compression error:', err));

}

const updateData = {

grade,

feedback,

feedbackVideoUrl,

gradedAt: new Date(),

rubricScores: rubricScores ? JSON.parse(rubricScores) : []

};

// Remove null fields

Object.keys(updateData).forEach(key => {

if (updateData[key] === null || updateData[key] === undefined) {

delete updateData[key];

}

});

const submission = await AssignmentSubmission.findByIdAndUpdate(

submissionId,

updateData,

{ new: true }

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

if (!submission) {

return res.status(404).json({

success: false,

message: "Submission not found"

});

}

res.status(200).json({

success: true,

message: "Grade submitted",

data: submission

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error grading submission",

error: error.message

});

}

}];

// Grant extension to a student

exports.grantExtension = async (req, res) => {

try {

const { submissionId } = req.params;

const { extensionDate } = req.body;

const submission = await AssignmentSubmission.findByIdAndUpdate(

submissionId,

{ extensionGranted: new Date(extensionDate) },

{ new: true }

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

if (!submission) {

return res.status(404).json({

success: false,

message: "Submission not found"

});

}

res.status(200).json({

success: true,

message: "Extension granted",

data: submission

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error granting extension",

error: error.message

});

}

};

// TEACHER fetches assignments they created

exports.getAssignmentsForTeacher = async (req, res) => {

try {

const assignments = await Assignment.find({

teacher: req.user.id,

school: req.user.schoolId

})

.populate('subject', 'subject\_name')

.populate('class', 'class\_text')

.sort({ createdAt: -1 });

res.status(200).json({

success: true,

data: assignments

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error fetching teacher assignments",

error: error.message

});

}

};

// Get assignment analytics

exports.getAssignmentAnalytics = async (req, res) => {

try {

const { assignmentId } = req.params;

const submissions = await AssignmentSubmission.find({

assignment: assignmentId

});

const totalSubmissions = submissions.length;

const gradedSubmissions = submissions.filter(s => s.grade !== undefined).length;

const lateSubmissions = submissions.filter(s => s.lateSubmission).length;

const grades = submissions

.filter(s => s.grade !== undefined)

.map(s => s.grade);

const averageGrade = grades.length > 0

? grades.reduce((a, b) => a + b, 0) / grades.length

: 0;

res.status(200).json({

success: true,

data: {

totalSubmissions,

gradedSubmissions,

lateSubmissions,

averageGrade,

gradeDistribution: grades

}

});

} catch (error) {

res.status(500).json({

success: false,

message: "Error fetching analytics",

error: error.message

});

}

};

const express = require('express');

const router = express.Router();

const assignmentController = require('../controllers/assignment.controller');

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

// Create a new assignment (TEACHER only)

router.post(

'/',

auth(['TEACHER']),

assignmentController.createAssignment

);

// Get single assignment

router.get(

'/:id',

auth(['TEACHER', 'STUDENT']),

assignmentController.getAssignment

);

// Update assignment (TEACHER only)

router.put(

'/:id',

auth(['TEACHER']),

assignmentController.updateAssignment

);

// Delete assignment (TEACHER only)

router.delete(

'/:id',

auth(['TEACHER']),

assignmentController.deleteAssignment

);

// Get assignments for student (STUDENT only)

router.get(

'/student/list',

auth(['STUDENT']),

assignmentController.getAssignmentsForStudent

);

// Student submits an assignment (STUDENT only)

router.post(

'/submit',

auth(['STUDENT']),

assignmentController.submitAssignment

);

// Get submissions for an assignment (TEACHER only)

router.get(

'/submissions/:assignmentId',

auth(['TEACHER','STUDENT']),

assignmentController.getSubmissionsForAssignment

);

// Grade a submission (TEACHER only)

router.post(

'/submissions/:submissionId/grade',

auth(['TEACHER']),

assignmentController.gradeSubmission

);

// Grant extension to a student (TEACHER only)

router.post(

'/submissions/:submissionId/extension',

auth(['TEACHER']),

assignmentController.grantExtension

);

// Get teacher's assignments (TEACHER only)

router.get(

'/teacher/list',

auth(['TEACHER']),

assignmentController.getAssignmentsForTeacher

);

// Get assignment analytics (TEACHER only)

router.get(

'/analytics/:assignmentId',

auth(['TEACHER']),

assignmentController.getAssignmentAnalytics

);

module.exports = router;

import React, { useEffect, useState, useMemo, useCallback } from "react";

import axios from "axios";

import { useForm } from "react-hook-form";

import { toast } from 'react-toastify';

import 'react-toastify/dist/ReactToastify.css';

import { keyframes, useTheme, alpha } from "@mui/material/styles";

import {

Box,

Typography,

IconButton,

Card,

CardContent,

Chip,

Avatar,

InputAdornment,

TextField,

Badge,

Stack,

Table,

TableBody,

TableCell,

TableContainer,

TableHead,

TableRow,

Paper,

Button,

Dialog,

DialogActions,

DialogContent,

DialogTitle,

CircularProgress,

Tooltip,

Divider,

MenuItem,

Select,

FormControl,

InputLabel,

LinearProgress,

Skeleton,

Grid,

FormGroup,

FormControlLabel,

Switch,

Accordion,

AccordionSummary,

AccordionDetails,

Tabs,

Tab,

useMediaQuery

} from "@mui/material";

import {

Download as DownloadIcon,

Delete as DeleteIcon,

Edit as EditIcon,

Visibility as VisibilityIcon,

CheckCircle as CheckCircleIcon,

Close as CloseIcon,

Search as SearchIcon,

FilterList as FilterListIcon,

Clear as ClearIcon,

Assignment as AssignmentIcon,

School as SchoolIcon,

Class as ClassIcon,

Subject as SubjectIcon,

CalendarToday as CalendarIcon,

Person as PersonIcon,

Email as EmailIcon,

ExpandMore as ExpandMoreIcon,

VideoFile as VideoIcon,

PlayCircle as PlayIcon,

BarChart as AnalyticsIcon,

Extension as ExtensionIcon,

Sort as SortIcon,

ArrowUpward as ArrowUpwardIcon,

ArrowDownward as ArrowDownwardIcon,

Info as InfoIcon,

Assessment as AssessmentIcon,

Grading as GradingIcon,

AccessTime as AccessTimeIcon

} from "@mui/icons-material";

const API\_BASE = "http://localhost:5000/api";

// Enhanced Animations

const fadeIn = keyframes`

from { opacity: 0; transform: translateY(10px); }

to { opacity: 1; transform: translateY(0); }

`;

const pulse = keyframes`

0% { transform: scale(1); }

50% { transform: scale(1.02); }

100% { transform: scale(1); }

`;

const gradientFlow = keyframes`

0% { background-position: 0% 50%; }

50% { background-position: 100% 50%; }

100% { background-position: 0% 50%; }

`;

const shimmer = keyframes`

0% { background-position: -200% 0; }

100% { background-position: 200% 0; }

`;

const TeacherAssignmentDashboard = () => {

const theme = useTheme();

const isMobile = useMediaQuery(theme.breakpoints.down('sm'));

const { register, handleSubmit, reset, setValue, watch, formState: { errors } } = useForm();

const [assignments, setAssignments] = useState([]);

const [selectedAssignmentId, setSelectedAssignmentId] = useState("");

const [submissions, setSubmissions] = useState([]);

const [loading, setLoading] = useState({

assignments: false,

submissions: false,

analytics: false,

form: false

});

const [classes, setClasses] = useState([]);

const [subjects, setSubjects] = useState([]);

const [isEditing, setIsEditing] = useState(false);

const [currentAssignment, setCurrentAssignment] = useState(null);

const [searchTerm, setSearchTerm] = useState("");

const [filterClass, setFilterClass] = useState("");

const [filterSubject, setFilterSubject] = useState("");

const [isSubmittingGrade, setIsSubmittingGrade] = useState(false);

const [gradeForm, setGradeForm] = useState({});

const [deleteDialogOpen, setDeleteDialogOpen] = useState(false);

const [assignmentToDelete, setAssignmentToDelete] = useState(null);

const [submissionDialogOpen, setSubmissionDialogOpen] = useState(false);

const [selectedSubmission, setSelectedSubmission] = useState(null);

const [analytics, setAnalytics] = useState(null);

const [extensionDialogOpen, setExtensionDialogOpen] = useState(false);

const [extensionDate, setExtensionDate] = useState("");

const [rubricScores, setRubricScores] = useState({});

const [sortConfig, setSortConfig] = useState({ key: 'dueDate', direction: 'asc' });

const [activeTab, setActiveTab] = useState('assignments');

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

const watchClass = watch("class");

// Memoized axios config

const axiosConfig = useMemo(() => ({

headers: { Authorization: `Bearer ${token}` },

withCredentials: true,

}), [token]);

// Formatting utilities

const formatDate = useCallback((dateString) => {

if (!dateString) return "Not scheduled";

const date = new Date(dateString);

return date.toLocaleString('en-GB', {

day: '2-digit',

month: 'short',

year: 'numeric',

hour: '2-digit',

minute: '2-digit'

});

}, []);

const formatDateForInput = useCallback((dateString) => {

if (!dateString) return "";

const date = new Date(dateString);

return date.toISOString().split('T')[0];

}, []);

const getTimeAgo = useCallback((dateString) => {

const date = new Date(dateString);

const now = new Date();

const diffInSeconds = Math.floor((now - date) / 1000);

if (diffInSeconds < 60) return 'Just now';

const diffInMinutes = Math.floor(diffInSeconds / 60);

if (diffInMinutes < 60) return `${diffInMinutes}m ago`;

const diffInHours = Math.floor(diffInMinutes / 60);

if (diffInHours < 24) return `${diffInHours}h ago`;

const diffInDays = Math.floor(diffInHours / 24);

if (diffInDays < 7) return `${diffInDays}d ago`;

return formatDate(dateString).split(',')[0];

}, [formatDate]);

// Data fetching functions

const fetchAssignments = useCallback(async () => {

try {

setLoading(prev => ({ ...prev, assignments: true }));

const response = await axios.get(

`${API\_BASE}/assignments/teacher/list`,

axiosConfig

);

setAssignments(response.data.data);

} catch (error) {

console.error("Error fetching assignments:", error);

toast.error("Failed to fetch assignments");

} finally {

setLoading(prev => ({ ...prev, assignments: false }));

}

}, [axiosConfig]);

const fetchClasses = useCallback(async () => {

try {

const response = await axios.get(`${API\_BASE}/class/all`, {

headers: {

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

},

});

setClasses(response.data.data);

} catch (error) {

console.error("Error fetching classes:", error);

toast.error("Failed to fetch classes");

}

}, []);

const fetchSubjects = useCallback(async () => {

try {

const response = await axios.get(`${API\_BASE}/subjects`, {

headers: {

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

},

});

setSubjects(response.data.data);

} catch (error) {

console.error("Error fetching subjects:", error);

toast.error("Failed to fetch subjects");

}

}, []);

const fetchSubmissions = useCallback(async (assignmentId) => {

try {

setLoading(prev => ({ ...prev, submissions: true }));

const response = await axios.get(

`${API\_BASE}/assignments/submissions/${assignmentId}`,

axiosConfig

);

setSubmissions(response.data.data);

} catch (error) {

console.error("Error fetching submissions:", error);

toast.error("Failed to fetch submissions");

} finally {

setLoading(prev => ({ ...prev, submissions: false }));

}

}, [axiosConfig]);

const fetchAnalytics = useCallback(async (assignmentId) => {

try {

setLoading(prev => ({ ...prev, analytics: true }));

const response = await axios.get(

`${API\_BASE}/assignments/analytics/${assignmentId}`,

axiosConfig

);

setAnalytics(response.data.data);

} catch (error) {

console.error("Error fetching analytics:", error);

toast.error("Failed to fetch assignment analytics");

} finally {

setLoading(prev => ({ ...prev, analytics: false }));

}

}, [axiosConfig]);

// Form submission handler

const onSubmit = useCallback(async (data) => {

try {

setLoading(prev => ({ ...prev, form: true }));

const formData = new FormData();

formData.append('title', data.title);

formData.append('description', data.description);

formData.append('subject', data.subject);

formData.append('class', data.class);

formData.append('dueDate', data.dueDate);

formData.append('videoUrl', data.videoUrl || '');

formData.append('rubric', JSON.stringify(data.rubric || []));

formData.append('maxPoints', data.maxPoints);

formData.append('allowLateSubmission', data.allowLateSubmission);

formData.append('peerReviewEnabled', data.peerReviewEnabled);

if (data.attachments) {

for (let i = 0; i < data.attachments.length; i++) {

formData.append('attachments', data.attachments[i]);

}

}

if (data.assignmentVideo) {

formData.append('assignmentVideo', data.assignmentVideo[0]);

}

const endpoint = isEditing && currentAssignment

? `${API\_BASE}/assignments/${currentAssignment.\_id}`

: `${API\_BASE}/assignments`;

const method = isEditing ? 'put' : 'post';

const response = await axios[method](endpoint, formData, {

...axiosConfig,

headers: {

...axiosConfig.headers,

'Content-Type': 'multipart/form-data'

}

});

toast.success(response.data.message);

reset();

fetchAssignments();

setIsEditing(false);

setCurrentAssignment(null);

} catch (error) {

console.error(error);

toast.error(error.response?.data?.message || "Operation failed");

} finally {

setLoading(prev => ({ ...prev, form: false }));

}

}, [isEditing, currentAssignment, axiosConfig, fetchAssignments, reset]);

// Assignment CRUD operations

const handleEdit = useCallback((assignment) => {

setIsEditing(true);

setCurrentAssignment(assignment);

setValue("title", assignment.title);

setValue("description", assignment.description);

setValue("subject", assignment.subject?.\_id);

setValue("class", assignment.class?.\_id);

setValue("dueDate", formatDateForInput(assignment.dueDate));

setValue("videoUrl", assignment.videoUrl || "");

setValue("maxPoints", assignment.maxPoints || 100);

setValue("allowLateSubmission", assignment.allowLateSubmission || false);

setValue("peerReviewEnabled", assignment.peerReviewEnabled || false);

setValue("rubric", assignment.rubric || []);

}, [setValue, formatDateForInput]);

const handleCancelEdit = useCallback(() => {

setIsEditing(false);

setCurrentAssignment(null);

reset();

}, [reset]);

const openDeleteDialog = useCallback((assignment) => {

setAssignmentToDelete(assignment);

setDeleteDialogOpen(true);

}, []);

const closeDeleteDialog = useCallback(() => {

setDeleteDialogOpen(false);

setAssignmentToDelete(null);

}, []);

const handleDelete = useCallback(async () => {

if (!assignmentToDelete) return;

try {

await axios.delete(

`${API\_BASE}/assignments/${assignmentToDelete.\_id}`,

axiosConfig

);

toast.success("Assignment deleted successfully");

fetchAssignments();

} catch (error) {

console.error(error);

toast.error("Failed to delete assignment");

} finally {

closeDeleteDialog();

}

}, [assignmentToDelete, axiosConfig, fetchAssignments, closeDeleteDialog]);

// Submission grading functions

const handleGradeChange = useCallback((submissionId, value) => {

setGradeForm(prev => ({

...prev,

[submissionId]: value

}));

}, []);

const handleRubricScoreChange = useCallback((submissionId, criteriaId, value) => {

setRubricScores(prev => ({

...prev,

[submissionId]: {

...(prev[submissionId] || {}),

[criteriaId]: value

}

}));

}, []);

const submitGrade = useCallback(async (submissionId) => {

const formData = new FormData();

if (gradeForm[submissionId]) {

formData.append('grade', gradeForm[submissionId]);

}

if (rubricScores[submissionId]) {

formData.append('rubricScores', JSON.stringify(

Object.entries(rubricScores[submissionId]).map(([criteriaId, score]) => ({

criteriaId,

score

}))

));

}

if (document.getElementById(`feedback-${submissionId}`).value) {

formData.append('feedback', document.getElementById(`feedback-${submissionId}`).value);

}

if (document.getElementById(`feedbackVideo-${submissionId}`).files[0]) {

formData.append('feedbackVideo', document.getElementById(`feedbackVideo-${submissionId}`).files[0]);

}

try {

setIsSubmittingGrade(true);

await axios.post(

`${API\_BASE}/assignments/submissions/${submissionId}/grade`,

formData,

{

...axiosConfig,

headers: {

...axiosConfig.headers,

'Content-Type': 'multipart/form-data'

}

}

);

toast.success("Grade submitted successfully");

fetchSubmissions(selectedAssignmentId);

setGradeForm(prev => ({ ...prev, [submissionId]: '' }));

} catch (error) {

console.error(error);

toast.error("Failed to submit grade");

} finally {

setIsSubmittingGrade(false);

}

}, [gradeForm, rubricScores, axiosConfig, selectedAssignmentId, fetchSubmissions]);

// Submission dialog functions

const openSubmissionDialog = useCallback((submission) => {

setSelectedSubmission(submission);

setSubmissionDialogOpen(true);

}, []);

const closeSubmissionDialog = useCallback(() => {

setSubmissionDialogOpen(false);

setSelectedSubmission(null);

}, []);

const openExtensionDialog = useCallback((submission) => {

setSelectedSubmission(submission);

setExtensionDialogOpen(true);

setExtensionDate("");

}, []);

const closeExtensionDialog = useCallback(() => {

setExtensionDialogOpen(false);

setSelectedSubmission(null);

setExtensionDate("");

}, []);

const grantExtension = useCallback(async () => {

if (!selectedSubmission || !extensionDate) return;

try {

await axios.post(

`${API\_BASE}/assignments/submissions/${selectedSubmission.\_id}/extension`,

{ extensionDate },

axiosConfig

);

toast.success("Extension granted successfully");

fetchSubmissions(selectedAssignmentId);

closeExtensionDialog();

} catch (error) {

console.error(error);

toast.error("Failed to grant extension");

}

}, [selectedSubmission, extensionDate, axiosConfig, selectedAssignmentId, fetchSubmissions, closeExtensionDialog]);

// Sorting functionality

const handleSort = useCallback((key) => {

let direction = 'asc';

if (sortConfig.key === key && sortConfig.direction === 'asc') {

direction = 'desc';

}

setSortConfig({ key, direction });

}, [sortConfig]);

// Filtered and sorted assignments

const filteredAssignments = useMemo(() => {

let result = assignments.filter(assignment => {

const matchesSearch = assignment.title.toLowerCase().includes(searchTerm.toLowerCase()) ||

assignment.description.toLowerCase().includes(searchTerm.toLowerCase());

const matchesClass = filterClass ? assignment.class?.\_id === filterClass : true;

const matchesSubject = filterSubject ? assignment.subject?.\_id === filterSubject : true;

return matchesSearch && matchesClass && matchesSubject;

});

// Sorting logic

if (sortConfig.key) {

result.sort((a, b) => {

const aValue = a[sortConfig.key] || '';

const bValue = b[sortConfig.key] || '';

if (sortConfig.key === 'dueDate') {

const dateA = new Date(a.dueDate).getTime();

const dateB = new Date(b.dueDate).getTime();

return sortConfig.direction === 'asc' ? dateA - dateB : dateB - dateA;

}

if (aValue < bValue) {

return sortConfig.direction === 'asc' ? -1 : 1;

}

if (aValue > bValue) {

return sortConfig.direction === 'asc' ? 1 : -1;

}

return 0;

});

}

return result;

}, [assignments, searchTerm, filterClass, filterSubject, sortConfig]);

const clearFilters = useCallback(() => {

setSearchTerm("");

setFilterClass("");

setFilterSubject("");

}, []);

// Skeleton loaders

const AssignmentSkeleton = useCallback(() => (

<Card

variant="outlined"

sx={{

borderRadius: 2,

overflow: 'hidden',

transition: 'all 0.3s ease',

animation: `${fadeIn} 0.5s ease-out`,

p: 2,

height: 100,

mb: 2

}}

>

<Box sx={{

height: '100%',

background: `linear-gradient(90deg, ${alpha(theme.palette.text.disabled, 0.1)} 25%,

${alpha(theme.palette.text.disabled, 0.05)} 50%,

${alpha(theme.palette.text.disabled, 0.1)} 75%)`,

backgroundSize: '200% 100%',

animation: `${shimmer} 1.5s infinite`,

borderRadius: 1

}} />

</Card>

), [theme]);

const SubmissionSkeleton = useCallback(() => (

<TableRow>

<TableCell>

<Box sx={{ display: 'flex', alignItems: 'center' }}>

<Skeleton variant="circular" width={40} height={40} />

<Box sx={{ ml: 2 }}>

<Skeleton variant="text" width={100} />

<Skeleton variant="text" width={150} />

</Box>

</Box>

</TableCell>

<TableCell><Skeleton variant="text" /></TableCell>

<TableCell><Skeleton variant="text" width={120} /></TableCell>

<TableCell><Skeleton variant="text" width={80} /></TableCell>

<TableCell><Skeleton variant="circular" width={24} height={24} /></TableCell>

</TableRow>

), []);

// Initial data fetch

useEffect(() => {

fetchAssignments();

fetchClasses();

fetchSubjects();

}, [fetchAssignments, fetchClasses, fetchSubjects]);

// Fetch submissions when assignment is selected

useEffect(() => {

if (selectedAssignmentId) {

fetchSubmissions(selectedAssignmentId);

fetchAnalytics(selectedAssignmentId);

}

}, [selectedAssignmentId, fetchSubmissions, fetchAnalytics]);

return (

<Box

sx={{

maxWidth: 'lg',

mx: "auto",

p: { xs: 1, sm: 3 },

animation: `${fadeIn} 0.5s ease-out`,

}}

>

{/\* Title Section \*/}

<Box sx={{

display: 'flex',

flexDirection: { xs: 'column', sm: 'row' },

justifyContent: 'space-between',

alignItems: { xs: 'flex-start', sm: 'center' },

mb: 4,

gap: 2

}}>

<Typography

variant="h3"

component="h1"

sx={{

background: `linear-gradient(45deg, ${theme.palette.primary.main} 30%, ${theme.palette.secondary.main} 90%)`,

WebkitBackgroundClip: "text",

WebkitTextFillColor: "transparent",

animation: `${gradientFlow} 6s ease infinite`,

backgroundSize: "200% 200%",

display: 'flex',

alignItems: 'center',

gap: 2,

fontSize: { xs: '1.8rem', sm: '2.4rem' }

}}

>

<AssignmentIcon fontSize="large" />

Assignment Dashboard

</Typography>

<Box sx={{ display: 'flex', alignItems: 'center', gap: 2, flexWrap: 'wrap' }}>

<Chip

icon={<CalendarIcon />}

label={`Last updated: ${formatDate(new Date()).split(',')[0]}`}

variant="outlined"

size="small"

sx={{ fontSize: { xs: '0.7rem', sm: '0.8125rem' } }}

/>

<Badge badgeContent={assignments.filter(a => new Date(a.dueDate) < new Date()).length} color="error">

<Chip

label={`Total: ${assignments.length}`}

variant="outlined"

avatar={<Avatar sx={{ width: 24, height: 24 }}>{assignments.length}</Avatar>}

color="primary"

sx={{ fontSize: { xs: '0.7rem', sm: '0.8125rem' } }}

/>

</Badge>

</Box>

</Box>

{/\* Navigation Tabs \*/}

<Box sx={{ borderBottom: 1, borderColor: 'divider', mb: 3 }}>

<Tabs

value={activeTab}

onChange={(e, newValue) => setActiveTab(newValue)}

variant="scrollable"

scrollButtons="auto"

>

<Tab

label="Assignments"

value="assignments"

icon={<AssignmentIcon />}

iconPosition="start"

sx={{ minHeight: 48 }}

/>

{selectedAssignmentId && (

<Tab

label="Submissions"

value="submissions"

icon={<SchoolIcon />}

iconPosition="start"

sx={{ minHeight: 48 }}

/>

)}

{analytics && (

<Tab

label="Analytics"

value="analytics"

icon={<AnalyticsIcon />}

iconPosition="start"

sx={{ minHeight: 48 }}

/>

)}

</Tabs>

</Box>

{/\* Create/Edit Assignment Form \*/}

{activeTab === 'assignments' && (

<Card

sx={{

mb: 4,

borderRadius: 2,

boxShadow: theme.shadows[2],

animation: isEditing ? `${pulse} 2s infinite` : 'none',

borderLeft: isEditing ? `4px solid ${theme.palette.primary.main}` : 'none'

}}

>

<CardContent>

<Typography variant="h5" component="h2" sx={{ mb: 2, display: 'flex', alignItems: 'center', gap: 1 }}>

{isEditing ? (

<>

<EditIcon color="primary" /> Edit Assignment

</>

) : (

<>

<AssignmentIcon color="primary" /> Create New Assignment

</>

)}

</Typography>

<Box

component="form"

onSubmit={handleSubmit(onSubmit)}

encType="multipart/form-data"

sx={{

display: 'grid',

gridTemplateColumns: { xs: '1fr', md: '1fr 1fr' },

gap: 3

}}

>

<TextField

{...register("title", { required: "Title is required" })}

label="Title"

placeholder="Assignment Title"

variant="outlined"

fullWidth

required

error={!!errors.title}

helperText={errors.title?.message}

sx={{

'& .MuiOutlinedInput-root': {

borderRadius: 2

}

}}

/>

<TextField

{...register("description")}

label="Description"

placeholder="Assignment Description"

variant="outlined"

fullWidth

multiline

rows={1}

sx={{

'& .MuiOutlinedInput-root': {

borderRadius: 2

}

}}

/>

<FormControl fullWidth error={!!errors.subject}>

<InputLabel>Subject \*</InputLabel>

<Select

{...register("subject", { required: "Subject is required" })}

label="Subject \*"

>

<MenuItem value=""><em>Select Subject</em></MenuItem>

{subjects.map((subject) => (

<MenuItem key={subject.\_id} value={subject.\_id}>

{subject.subject\_name}

</MenuItem>

))}

</Select>

{errors.subject && (

<Typography variant="caption" color="error">

{errors.subject.message}

</Typography>

)}

</FormControl>

<FormControl fullWidth error={!!errors.class}>

<InputLabel>Class \*</InputLabel>

<Select

{...register("class", { required: "Class is required" })}

label="Class \*"

>

<MenuItem value=""><em>Select Class</em></MenuItem>

{classes.map((cls) => (

<MenuItem key={cls.\_id} value={cls.\_id}>

{cls.class\_text}

</MenuItem>

))}

</Select>

{errors.class && (

<Typography variant="caption" color="error">

{errors.class.message}

</Typography>

)}

</FormControl>

<TextField

{...register("dueDate", { required: "Due date is required" })}

label="Due Date \*"

type="date"

InputLabelProps={{ shrink: true }}

variant="outlined"

fullWidth

error={!!errors.dueDate}

helperText={errors.dueDate?.message}

inputProps={{

min: new Date().toISOString().split('T')[0]

}}

sx={{

'& .MuiOutlinedInput-root': {

borderRadius: 2

}

}}

/>

<TextField

{...register("maxPoints", {

valueAsNumber: true,

min: { value: 1, message: "Minimum 1 point" },

max: { value: 1000, message: "Maximum 1000 points" }

})}

label="Max Points"

type="number"

variant="outlined"

fullWidth

defaultValue={100}

error={!!errors.maxPoints}

helperText={errors.maxPoints?.message}

InputProps={{ inputProps: { min: 1, max: 1000 } }}

sx={{

'& .MuiOutlinedInput-root': {

borderRadius: 2

}

}}

/>

<Box sx={{ gridColumn: '1 / -1' }}>

<FormGroup>

<FormControlLabel

control={<Switch {...register("allowLateSubmission")} />}

label="Allow Late Submissions"

/>

<FormControlLabel

control={<Switch {...register("peerReviewEnabled")} />}

label="Enable Peer Review"

/>

</FormGroup>

</Box>

<Box sx={{ gridColumn: '1 / -1' }}>

<Typography variant="subtitle1" gutterBottom>

Assignment Video

</Typography>

<input

type="file"

accept="video/\*"

{...register("assignmentVideo")}

/>

<TextField

{...register("videoUrl")}

label="Or Video URL"

placeholder="https://example.com/video.mp4"

variant="outlined"

fullWidth

sx={{ mt: 1 }}

/>

</Box>

<Box sx={{ gridColumn: '1 / -1' }}>

<Typography variant="subtitle1" gutterBottom>

Attachments

</Typography>

<input

type="file"

multiple

{...register("attachments")}

/>

</Box>

{/\* <Box sx={{ gridColumn: '1 / -1' }}>

<Accordion>

<AccordionSummary expandIcon={<ExpandMoreIcon />}>

<Typography>Rubric Settings</Typography>

</AccordionSummary>

<AccordionDetails>

<Typography variant="body2" color="text.secondary">

Advanced rubric builder coming soon

</Typography>

</AccordionDetails>

</Accordion>

</Box> \*/}

<Box

sx={{

gridColumn: { xs: '1', md: '1 / -1' },

display: 'flex',

gap: 2,

justifyContent: 'flex-end'

}}

>

{isEditing && (

<Button

variant="outlined"

color="secondary"

onClick={handleCancelEdit}

sx={{ borderRadius: 2 }}

>

Cancel

</Button>

)}

<Button

type="submit"

variant="contained"

color="primary"

disabled={loading.form}

sx={{ borderRadius: 2 }}

startIcon={loading.form ? <CircularProgress size={20} color="inherit" /> : null}

>

{isEditing ? "Update Assignment" : "Create Assignment"}

</Button>

</Box>

</Box>

</CardContent>

</Card>

)}

{/\* Filter/Search Section \*/}

{activeTab === 'assignments' && (

<Card sx={{ mb: 4, borderRadius: 2, boxShadow: theme.shadows[1] }}>

<CardContent>

<Typography variant="h5" component="h2" sx={{ mb: 2, display: 'flex', alignItems: 'center', gap: 1 }}>

<FilterListIcon color="primary" /> Filter Assignments

</Typography>

<Box

sx={{

display: 'grid',

gridTemplateColumns: { xs: '1fr', sm: '1fr 1fr', md: '2fr 1fr 1fr auto' },

gap: 2,

alignItems: 'flex-end'

}}

>

<TextField

label="Search"

placeholder="Search assignments..."

variant="outlined"

value={searchTerm}

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

InputProps={{

startAdornment: (

<InputAdornment position="start">

<SearchIcon />

</InputAdornment>

),

endAdornment: searchTerm && (

<InputAdornment position="end">

<IconButton size="small" onClick={() => setSearchTerm("")}>

<CloseIcon fontSize="small" />

</IconButton>

</InputAdornment>

),

}}

fullWidth

sx={{

'& .MuiOutlinedInput-root': {

borderRadius: 2

}

}}

/>

<FormControl fullWidth>

<InputLabel>Class</InputLabel>

<Select

value={filterClass}

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

label="Class"

>

<MenuItem value="">All Classes</MenuItem>

{classes.map((cls) => (

<MenuItem key={cls.\_id} value={cls.\_id}>

{cls.class\_text}

</MenuItem>

))}

</Select>

</FormControl>

<FormControl fullWidth>

<InputLabel>Subject</InputLabel>

<Select

value={filterSubject}

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

label="Subject"

>

<MenuItem value="">All Subjects</MenuItem>

{subjects.map((subject) => (

<MenuItem key={subject.\_id} value={subject.\_id}>

{subject.subject\_name}

</MenuItem>

))}

</Select>

</FormControl>

<Button

variant="outlined"

onClick={clearFilters}

startIcon={<ClearIcon />}

sx={{ height: '56px', borderRadius: 2 }}

>

Clear

</Button>

</Box>

</CardContent>

</Card>

)}

{/\* Assignments List \*/}

{activeTab === 'assignments' && (

<Card sx={{ mb: 4, borderRadius: 2, boxShadow: theme.shadows[1] }}>

<CardContent>

<Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 2 }}>

<Typography variant="h5" component="h2" sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<AssignmentIcon color="primary" /> Your Assignments

</Typography>

<Typography variant="body2" color="text.secondary">

Showing {filteredAssignments.length} of {assignments.length} assignments

</Typography>

</Box>

{loading.assignments && assignments.length === 0 ? (

<Box>

{[1, 2, 3].map((i) => (

<AssignmentSkeleton key={i} />

))}

</Box>

) : filteredAssignments.length === 0 ? (

<Box

sx={{

textAlign: 'center',

p: 4,

borderRadius: 2,

bgcolor: theme.palette.action.hover,

}}

>

<AssignmentIcon sx={{ fontSize: 60, color: 'text.disabled', mb: 2 }} />

<Typography variant="h6" color="text.secondary" gutterBottom>

No assignments found

</Typography>

<Typography variant="body2" color="text.secondary">

{searchTerm || filterClass || filterSubject

? "No assignments match your current filters"

: "You haven't created any assignments yet"}

</Typography>

</Box>

) : (

<TableContainer component={Paper} elevation={0} sx={{ borderRadius: 2 }}>

<Table>

<TableHead sx={{ bgcolor: theme.palette.grey[100] }}>

<TableRow>

<TableCell>

<Button

onClick={() => handleSort('title')}

startIcon={<SortIcon />}

endIcon={

sortConfig.key === 'title' ? (

sortConfig.direction === 'asc' ? <ArrowUpwardIcon fontSize="small" /> : <ArrowDownwardIcon fontSize="small" />

) : null

}

sx={{ textTransform: 'none', fontWeight: 'bold' }}

>

Title

</Button>

</TableCell>

<TableCell>

<Button

onClick={() => handleSort('subject')}

startIcon={<SortIcon />}

endIcon={

sortConfig.key === 'subject' ? (

sortConfig.direction === 'asc' ? <ArrowUpwardIcon fontSize="small" /> : <ArrowDownwardIcon fontSize="small" />

) : null

}

sx={{ textTransform: 'none', fontWeight: 'bold' }}

>

Subject

</Button>

</TableCell>

<TableCell>

<Button

onClick={() => handleSort('class')}

startIcon={<SortIcon />}

endIcon={

sortConfig.key === 'class' ? (

sortConfig.direction === 'asc' ? <ArrowUpwardIcon fontSize="small" /> : <ArrowDownwardIcon fontSize="small" />

) : null

}

sx={{ textTransform: 'none', fontWeight: 'bold' }}

>

Class

</Button>

</TableCell>

<TableCell>

<Button

onClick={() => handleSort('dueDate')}

startIcon={<SortIcon />}

endIcon={

sortConfig.key === 'dueDate' ? (

sortConfig.direction === 'asc' ? <ArrowUpwardIcon fontSize="small" /> : <ArrowDownwardIcon fontSize="small" />

) : null

}

sx={{ textTransform: 'none', fontWeight: 'bold' }}

>

Due Date

</Button>

</TableCell>

<TableCell>Submissions</TableCell>

<TableCell align="right">Actions</TableCell>

</TableRow>

</TableHead>

<TableBody>

{filteredAssignments.map((assignment) => (

<TableRow

key={assignment.\_id}

hover

sx={{

'&:last-child td': { borderBottom: 0 },

animation: `${fadeIn} 0.3s ease-out`,

animationDelay: `${filteredAssignments.indexOf(assignment) \* 50}ms`

}}

>

<TableCell>

<Typography fontWeight="medium">{assignment.title}</Typography>

<Typography variant="body2" color="text.secondary" noWrap>

{assignment.description}

</Typography>

{assignment.videoUrl && (

<Chip

icon={<VideoIcon />}

label="Video"

size="small"

color="info"

sx={{ mt: 0.5 }}

/>

)}

</TableCell>

<TableCell>

<Chip

icon={<SubjectIcon />}

label={assignment.subject?.subject\_name || "N/A"}

size="small"

sx={{ bgcolor: theme.palette.primary.light }}

/>

</TableCell>

<TableCell>

<Chip

icon={<ClassIcon />}

label={assignment.class?.class\_text || "N/A"}

size="small"

sx={{ bgcolor: theme.palette.secondary.light }}

/>

</TableCell>

<TableCell>

<Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<CalendarIcon fontSize="small" color="action" />

<Typography variant="body2">

{formatDate(assignment.dueDate)}

</Typography>

{new Date(assignment.dueDate) < new Date() && (

<Chip

label="Overdue"

size="small"

color="error"

sx={{ ml: 1 }}

/>

)}

</Box>

</TableCell>

<TableCell>

<Button

variant="text"

color="primary"

onClick={() => {

setSelectedAssignmentId(assignment.\_id === selectedAssignmentId ? "" : assignment.\_id);

setActiveTab('submissions');

if (assignment.\_id !== selectedAssignmentId) {

fetchSubmissions(assignment.\_id);

}

}}

startIcon={<SchoolIcon />}

sx={{ textTransform: 'none' }}

>

{assignment.submissionCount || 0} submissions

</Button>

</TableCell>

<TableCell align="right">

<Box sx={{ display: 'flex', gap: 1, justifyContent: 'flex-end' }}>

<Tooltip title="Edit">

<IconButton

onClick={() => handleEdit(assignment)}

color="primary"

>

<EditIcon />

</IconButton>

</Tooltip>

<Tooltip title="Delete">

<IconButton

onClick={() => openDeleteDialog(assignment)}

color="error"

>

<DeleteIcon />

</IconButton>

</Tooltip>

<Tooltip title="Analytics">

<IconButton

onClick={() => {

setSelectedAssignmentId(assignment.\_id);

setActiveTab('analytics');

fetchAnalytics(assignment.\_id);

}}

color="info"

>

<AnalyticsIcon />

</IconButton>

</Tooltip>

</Box>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

)}

</CardContent>

</Card>

)}

{/\* Analytics Section \*/}

{activeTab === 'analytics' && analytics && selectedAssignmentId && (

<Card sx={{ mb: 4, borderRadius: 2, boxShadow: theme.shadows[1] }}>

<CardContent>

<Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 2 }}>

<Typography variant="h5" component="h2" sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<AnalyticsIcon color="primary" /> Assignment Analytics

</Typography>

<Box>

<Button

variant="outlined"

onClick={() => setActiveTab('submissions')}

startIcon={<SchoolIcon />}

sx={{ mr: 1 }}

>

View Submissions

</Button>

<IconButton onClick={() => setAnalytics(null)}>

<CloseIcon />

</IconButton>

</Box>

</Box>

<Grid container spacing={3}>

<Grid item xs={12} sm={6} md={3}>

<Card sx={{

textAlign: 'center',

p: 2,

borderLeft: `4px solid ${theme.palette.primary.main}`,

height: '100%'

}}>

<Typography variant="h4" color="primary">

{analytics.totalSubmissions}

</Typography>

<Typography variant="body2">Total Submissions</Typography>

<Typography variant="caption" color="text.secondary">

{analytics.submissionRate}% submission rate

</Typography>

</Card>

</Grid>

<Grid item xs={12} sm={6} md={3}>

<Card sx={{

textAlign: 'center',

p: 2,

borderLeft: `4px solid ${theme.palette.success.main}`,

height: '100%'

}}>

<Typography variant="h4" color="success.main">

{analytics.gradedSubmissions}

</Typography>

<Typography variant="body2">Graded</Typography>

<Typography variant="caption" color="text.secondary">

{analytics.gradedPercentage}% of submissions

</Typography>

</Card>

</Grid>

<Grid item xs={12} sm={6} md={3}>

<Card sx={{

textAlign: 'center',

p: 2,

borderLeft: `4px solid ${theme.palette.error.main}`,

height: '100%'

}}>

<Typography variant="h4" color="error.main">

{analytics.lateSubmissions}

</Typography>

<Typography variant="body2">Late</Typography>

<Typography variant="caption" color="text.secondary">

{analytics.latePercentage}% of submissions

</Typography>

</Card>

</Grid>

<Grid item xs={12} sm={6} md={3}>

<Card sx={{

textAlign: 'center',

p: 2,

borderLeft: `4px solid ${theme.palette.warning.main}`,

height: '100%'

}}>

<Typography variant="h4">

{analytics.averageGrade.toFixed(1)}%

</Typography>

<Typography variant="body2">Avg Grade</Typography>

<Typography variant="caption" color="text.secondary">

{analytics.passingRate}% passing rate

</Typography>

</Card>

</Grid>

<Grid item xs={12}>

<Card sx={{ p: 2 }}>

<Typography variant="h6" gutterBottom sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<AssessmentIcon /> Grade Distribution

</Typography>

<Box sx={{ width: '100%', height: 200, p: 2 }}>

{/\* Placeholder for chart \*/}

<Box sx={{

display: 'flex',

alignItems: 'flex-end',

height: '100%',

gap: 1,

justifyContent: 'center'

}}>

{[70, 50, 30, 20, 10, 5].map((value, index) => (

<Box key={index} sx={{

width: 40,

height: `${value}%`,

bgcolor: theme.palette.primary.main,

borderRadius: 1,

display: 'flex',

alignItems: 'flex-end',

justifyContent: 'center',

color: 'white',

fontWeight: 'bold'

}}>

{value}%

</Box>

))}

</Box>

</Box>

</Card>

</Grid>

<Grid item xs={12} md={6}>

<Card sx={{ p: 2 }}>

<Typography variant="h6" gutterBottom sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<GradingIcon /> Submission Timeline

</Typography>

<Box sx={{ width: '100%', height: 200, p: 2 }}>

{/\* Placeholder for timeline \*/}

<Box sx={{

height: '100%',

display: 'flex',

alignItems: 'center',

justifyContent: 'center',

bgcolor: theme.palette.action.hover,

borderRadius: 1

}}>

<Typography variant="body2" color="text.secondary">

Submission timeline visualization

</Typography>

</Box>

</Box>

</Card>

</Grid>

<Grid item xs={12} md={6}>

<Card sx={{ p: 2 }}>

<Typography variant="h6" gutterBottom sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<AccessTimeIcon /> Time Spent Analysis

</Typography>

<Box sx={{ width: '100%', height: 200, p: 2 }}>

{/\* Placeholder for time spent \*/}

<Box sx={{

height: '100%',

display: 'flex',

alignItems: 'center',

justifyContent: 'center',

bgcolor: theme.palette.action.hover,

borderRadius: 1

}}>

<Typography variant="body2" color="text.secondary">

Average time spent: 45 minutes

</Typography>

</Box>

</Box>

</Card>

</Grid>

</Grid>

</CardContent>

</Card>

)}

{/\* Submissions Section \*/}

{activeTab === 'submissions' && selectedAssignmentId && (

<Card sx={{ borderRadius: 2, boxShadow: theme.shadows[1] }}>

<CardContent>

<Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 2 }}>

<Typography variant="h5" component="h2" sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<SchoolIcon color="primary" /> Submissions

</Typography>

<Box sx={{ display: 'flex', gap: 1 }}>

<Button

variant="outlined"

onClick={() => setActiveTab('analytics')}

startIcon={<AnalyticsIcon />}

>

View Analytics

</Button>

<IconButton onClick={() => setSelectedAssignmentId("")}>

<CloseIcon />

</IconButton>

</Box>

</Box>

{loading.submissions && submissions.length === 0 ? (

<Box>

{[1, 2, 3, 4, 5].map((i) => (

<SubmissionSkeleton key={i} />

))}

</Box>

) : submissions.length === 0 ? (

<Box

sx={{

textAlign: 'center',

p: 4,

borderRadius: 2,

bgcolor: theme.palette.action.hover,

}}

>

<SchoolIcon sx={{ fontSize: 60, color: 'text.disabled', mb: 2 }} />

<Typography variant="h6" color="text.secondary" gutterBottom>

No submissions yet

</Typography>

<Typography variant="body2" color="text.secondary">

Students haven't submitted any work for this assignment yet

</Typography>

</Box>

) : (

<TableContainer component={Paper} elevation={0} sx={{ borderRadius: 2 }}>

<Table>

<TableHead sx={{ bgcolor: theme.palette.grey[100] }}>

<TableRow>

<TableCell>Student</TableCell>

<TableCell>Submission</TableCell>

<TableCell>Submitted On</TableCell>

<TableCell>Grade</TableCell>

<TableCell align="right">Actions</TableCell>

</TableRow>

</TableHead>

<TableBody>

{submissions.map((sub) => (

<TableRow

key={sub.\_id}

hover

sx={{

'&:last-child td': { borderBottom: 0 },

animation: `${fadeIn} 0.3s ease-out`,

animationDelay: `${submissions.indexOf(sub) \* 50}ms`

}}

>

<TableCell>

<Box sx={{ display: 'flex', alignItems: 'center', gap: 2 }}>

<Avatar

src={sub.student?.image\_url}

sx={{ width: 40, height: 40 }}

>

{sub.student?.name?.charAt(0) || '?'}

</Avatar>

<Box>

<Typography fontWeight="medium">

{sub.student?.name || 'Unknown'}

</Typography>

<Typography variant="body2" color="text.secondary" sx={{ display: 'flex', alignItems: 'center', gap: 0.5 }}>

<EmailIcon fontSize="small" />

{sub.student?.email || 'N/A'}

</Typography>

{sub.lateSubmission && (

<Chip

label="Late"

size="small"

color="error"

sx={{ mt: 0.5 }}

/>

)}

</Box>

</Box>

</TableCell>

<TableCell>

<Box sx={{ display: 'flex', gap: 1, flexWrap: 'wrap' }}>

{sub.fileUrl && (

<Button

variant="outlined"

startIcon={<DownloadIcon />}

onClick={() => window.open(sub.fileUrl, '\_blank')}

sx={{ textTransform: 'none' }}

>

Document

</Button>

)}

{sub.videoUrl && (

<Button

variant="outlined"

startIcon={<PlayIcon />}

onClick={() => window.open(sub.videoUrl, '\_blank')}

sx={{ textTransform: 'none' }}

>

Video

</Button>

)}

</Box>

{sub.remarks && (

<Typography variant="body2" color="text.secondary" sx={{ mt: 1 }}>

Remarks: {sub.remarks}

</Typography>

)}

</TableCell>

<TableCell>

<Typography variant="body2">

{formatDate(sub.createdAt)}

</Typography>

<Typography variant="caption" color="text.secondary">

({getTimeAgo(sub.createdAt)})

</Typography>

</TableCell>

<TableCell>

{sub.grade !== undefined ? (

<Box>

<Chip

label={`${sub.grade}%`}

color={sub.grade >= 70 ? 'success' : sub.grade >= 50 ? 'warning' : 'error'}

variant="outlined"

sx={{ fontWeight: 'bold' }}

/>

{sub.feedback && (

<Tooltip title={sub.feedback}>

<InfoIcon color="action" sx={{ ml: 1, verticalAlign: 'middle' }} />

</Tooltip>

)}

</Box>

) : (

<Box>

<TextField

size="small"

placeholder="Grade"

value={gradeForm[sub.\_id] || ''}

onChange={(e) => handleGradeChange(sub.\_id, e.target.value)}

sx={{ width: 80, mb: 1 }}

type="number"

inputProps={{ min: 0, max: 100 }}

/>

<TextField

id={`feedback-${sub.\_id}`}

placeholder="Feedback"

size="small"

multiline

rows={2}

sx={{ width: '100%', mb: 1 }}

/>

<input

id={`feedbackVideo-${sub.\_id}`}

type="file"

accept="video/\*"

style={{ display: 'none' }}

/>

<label htmlFor={`feedbackVideo-${sub.\_id}`}>

<Button

variant="outlined"

size="small"

component="span"

startIcon={<VideoIcon />}

sx={{ mb: 1 }}

>

Video Feedback

</Button>

</label>

<Button

variant="contained"

size="small"

onClick={() => submitGrade(sub.\_id)}

disabled={isSubmittingGrade}

sx={{ ml: 1 }}

>

{isSubmittingGrade ? <CircularProgress size={20} /> : 'Submit'}

</Button>

</Box>

)}

</TableCell>

<TableCell align="right">

<Box sx={{ display: 'flex', gap: 0.5 }}>

<Tooltip title="View Details">

<IconButton

onClick={() => openSubmissionDialog(sub)}

color="primary"

>

<VisibilityIcon />

</IconButton>

</Tooltip>

<Tooltip title="Grant Extension">

<IconButton

onClick={() => openExtensionDialog(sub)}

color="secondary"

disabled={sub.grade !== undefined}

>

<ExtensionIcon />

</IconButton>

</Tooltip>

</Box>

</TableCell>

</TableRow>

))}

</TableBody>

</Table>

</TableContainer>

)}

</CardContent>

</Card>

)}

{/\* Delete Confirmation Dialog \*/}

<Dialog open={deleteDialogOpen} onClose={closeDeleteDialog}>

<DialogTitle>

Confirm Delete

</DialogTitle>

<DialogContent>

<Typography>

Are you sure you want to delete the assignment "{assignmentToDelete?.title}"?

</Typography>

<Typography variant="body2" color="text.secondary" sx={{ mt: 1 }}>

This action cannot be undone and will also delete all associated submissions.

</Typography>

</DialogContent>

<DialogActions>

<Button onClick={closeDeleteDialog}>Cancel</Button>

<Button onClick={handleDelete} color="error" variant="contained">

Delete

</Button>

</DialogActions>

</Dialog>

{/\* Submission Details Dialog \*/}

<Dialog

open={submissionDialogOpen}

onClose={closeSubmissionDialog}

maxWidth="md"

fullWidth

fullScreen={isMobile}

>

<DialogTitle sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<VisibilityIcon color="primary" /> Submission Details

</DialogTitle>

<DialogContent>

{selectedSubmission && (

<Box sx={{ mt: 2 }}>

<Box sx={{ display: 'flex', alignItems: 'center', gap: 3, mb: 3, flexDirection: { xs: 'column', sm: 'row' } }}>

<Avatar

src={selectedSubmission.student?.image\_url}

sx={{ width: 64, height: 64 }}

>

{selectedSubmission.student?.name?.charAt(0) || '?'}

</Avatar>

<Box sx={{ textAlign: { xs: 'center', sm: 'left' } }}>

<Typography variant="h6" fontWeight="medium">

{selectedSubmission.student?.name || 'Unknown Student'}

</Typography>

<Typography variant="body2" color="text.secondary">

{selectedSubmission.student?.email || 'N/A'}

</Typography>

{selectedSubmission.grade !== undefined && (

<Chip

label={`Grade: ${selectedSubmission.grade}%`}

color={selectedSubmission.grade >= 70 ? 'success' : selectedSubmission.grade >= 50 ? 'warning' : 'error'}

sx={{ mt: 1, fontWeight: 'bold' }}

/>

)}

</Box>

</Box>

<Divider sx={{ my: 2 }} />

<Box sx={{ mb: 3 }}>

<Typography variant="subtitle1" fontWeight="medium" gutterBottom>

Assignment Details

</Typography>

<Typography>

<strong>Title:</strong> {assignments.find(a => a.\_id === selectedSubmission.assignment)?.title || 'N/A'}

</Typography>

<Typography>

<strong>Submitted On:</strong> {formatDate(selectedSubmission.createdAt)}

</Typography>

{selectedSubmission.lateSubmission && (

<Typography color="error">

<strong>Late Submission</strong>

</Typography>

)}

{selectedSubmission.remarks && (

<Typography sx={{ mt: 1 }}>

<strong>Remarks:</strong> {selectedSubmission.remarks}

</Typography>

)}

</Box>

<Divider sx={{ my: 2 }} />

<Box sx={{ mb: 3 }}>

<Typography variant="subtitle1" fontWeight="medium" gutterBottom>

Submission Files

</Typography>

<Box sx={{ display: 'flex', gap: 2, flexWrap: 'wrap', justifyContent: { xs: 'center', sm: 'flex-start' } }}>

{selectedSubmission.fileUrl && (

<Button

variant="contained"

startIcon={<DownloadIcon />}

onClick={() => window.open(selectedSubmission.fileUrl, '\_blank')}

sx={{ borderRadius: 2 }}

>

Download Document

</Button>

)}

{selectedSubmission.videoUrl && (

<Button

variant="contained"

startIcon={<PlayIcon />}

onClick={() => window.open(selectedSubmission.videoUrl, '\_blank')}

sx={{ borderRadius: 2 }}

>

Play Submission Video

</Button>

)}

</Box>

</Box>

{selectedSubmission.feedback && (

<>

<Divider sx={{ my: 2 }} />

<Box>

<Typography variant="subtitle1" fontWeight="medium" gutterBottom>

Teacher Feedback

</Typography>

<Card variant="outlined" sx={{ p: 2, mb: 2, bgcolor: theme.palette.action.hover }}>

<Typography>

{selectedSubmission.feedback}

</Typography>

</Card>

{selectedSubmission.feedbackVideoUrl && (

<Button

variant="contained"

startIcon={<PlayIcon />}

onClick={() => window.open(selectedSubmission.feedbackVideoUrl, '\_blank')}

sx={{ borderRadius: 2 }}

>

Play Feedback Video

</Button>

)}

</Box>

</>

)}

</Box>

)}

</DialogContent>

<DialogActions>

<Button onClick={closeSubmissionDialog}>Close</Button>

</DialogActions>

</Dialog>

{/\* Extension Dialog \*/}

<Dialog open={extensionDialogOpen} onClose={closeExtensionDialog}>

<DialogTitle>

<Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>

<ExtensionIcon color="primary" /> Grant Extension

</Box>

</DialogTitle>

<DialogContent>

{selectedSubmission && (

<Box sx={{ mt: 2, minWidth: { xs: 'auto', sm: 400 } }}>

<Typography gutterBottom>

Grant extension for: <strong>{selectedSubmission.student?.name}</strong>

</Typography>

<TextField

label="New Due Date"

type="datetime-local"

InputLabelProps={{ shrink: true }}

variant="outlined"

fullWidth

value={extensionDate}

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

sx={{ mt: 2 }}

inputProps={{

min: new Date().toISOString().slice(0, 16)

}}

/>

<Typography variant="body2" color="text.secondary" sx={{ mt: 1 }}>

The student will have until this date to submit their work.

</Typography>

</Box>

)}

</DialogContent>

<DialogActions>

<Button onClick={closeExtensionDialog}>Cancel</Button>

<Button

onClick={grantExtension}

variant="contained"

disabled={!extensionDate}

>

Grant Extension

</Button>

</DialogActions>

</Dialog>

</Box>

);

};

export default TeacherAssignmentDashboard;

import \* as React from "react";

import { styled, useTheme } from "@mui/material/styles";

import Box from "@mui/material/Box";

import MuiDrawer from "@mui/material/Drawer";

import MuiAppBar from "@mui/material/AppBar";

import Toolbar from "@mui/material/Toolbar";

import List from "@mui/material/List";

import CssBaseline from "@mui/material/CssBaseline";

import Typography from "@mui/material/Typography";

import Divider from "@mui/material/Divider";

import IconButton from "@mui/material/IconButton";

import MenuIcon from "@mui/icons-material/Menu";

import ChevronLeftIcon from "@mui/icons-material/ChevronLeft";

import ChevronRightIcon from "@mui/icons-material/ChevronRight";

import ListItem from "@mui/material/ListItem";

import ListItemButton from "@mui/material/ListItemButton";

import ListItemIcon from "@mui/material/ListItemIcon";

import ListItemText from "@mui/material/ListItemText";

import InboxIcon from "@mui/icons-material/MoveToInbox";

import MailIcon from "@mui/icons-material/Mail";

import { Outlet, useNavigate } from "react-router-dom";

import DashboardCustomizeIcon from "@mui/icons-material/DashboardCustomize"; //dashboardcustomize icon

import PeopleIcon from "@mui/icons-material/People"; //student

import PeopleAltIcon from "@mui/icons-material/PeopleAlt"; //teacher

import EventIcon from "@mui/icons-material/Event"; //schedule

import NotificationsIcon from "@mui/icons-material/Notifications"; //notification

import SubjectIcon from "@mui/icons-material/Subject"; //subject

import ExplicitIcon from "@mui/icons-material/Explicit"; //examinations

import RecentActorsIcon from "@mui/icons-material/RecentActors"; //attendance

import LocalLibraryIcon from "@mui/icons-material/LocalLibrary"; //class

import HomeIcon from "@mui/icons-material/Home";

import LogoutIcon from "@mui/icons-material/Logout";

import AssignmentIcon from '@mui/icons-material/Assignment';

const drawerWidth = 240;

const openedMixin = (theme) => ({

width: drawerWidth,

transition: theme.transitions.create("width", {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.enteringScreen,

}),

overflowX: "hidden",

});

const closedMixin = (theme) => ({

transition: theme.transitions.create("width", {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.leavingScreen,

}),

overflowX: "hidden",

width: `calc(${theme.spacing(7)} + 1px)`,

[theme.breakpoints.up("sm")]: {

width: `calc(${theme.spacing(8)} + 1px)`,

},

});

const DrawerHeader = styled("div")(({ theme }) => ({

display: "flex",

alignItems: "center",

justifyContent: "flex-end",

padding: theme.spacing(0, 1),

// necessary for content to be below app bar

...theme.mixins.toolbar,

}));

const AppBar = styled(MuiAppBar, {

shouldForwardProp: (prop) => prop !== "open",

})(({ theme }) => ({

zIndex: theme.zIndex.drawer + 1,

transition: theme.transitions.create(["width", "margin"], {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.leavingScreen,

}),

variants: [

{

props: ({ open }) => open,

style: {

marginLeft: drawerWidth,

width: `calc(100% - ${drawerWidth}px)`,

transition: theme.transitions.create(["width", "margin"], {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.enteringScreen,

}),

},

},

],

}));

const Drawer = styled(MuiDrawer, {

shouldForwardProp: (prop) => prop !== "open",

})(({ theme }) => ({

width: drawerWidth,

flexShrink: 0,

whiteSpace: "nowrap",

boxSizing: "border-box",

variants: [

{

props: ({ open }) => open,

style: {

...openedMixin(theme),

"& .MuiDrawer-paper": openedMixin(theme),

},

},

{

props: ({ open }) => !open,

style: {

...closedMixin(theme),

"& .MuiDrawer-paper": closedMixin(theme),

},

},

],

}));

export default function Student() {

const theme = useTheme();

const [open, setOpen] = React.useState(false);

const handleDrawerOpen = () => {

setOpen(true);

};

const handleDrawerClose = () => {

setOpen(false);

};

const navArr = [

{ link: "/", component: "Home", icon: HomeIcon },

{

link: "/student",

component: "Your Details",

icon: DashboardCustomizeIcon,

},

{ link: "/student/schedule", component: "Schedule", icon: EventIcon },

{

link: "/student/attendance",

component: "Attendance",

icon: RecentActorsIcon,

},

{ link: "/student/assignment", component: "Assignment", icon: AssignmentIcon },

{

link: "/student/examinations",

component: "Examinations",

icon: ExplicitIcon,

},

{ link: "/student/notice", component: "Notice", icon: NotificationsIcon },

{ link: "/logout", component: "Logout", icon: LogoutIcon },

];

const navigate = useNavigate();

const handleNavigation = (link) => {

navigate(link);

};

return (

<Box sx={{ display: "flex" }}>

<CssBaseline />

<AppBar position="fixed" open={open}>

<Toolbar>

<IconButton

color="inherit"

aria-label="open drawer"

onClick={handleDrawerOpen}

edge="start"

sx={[

{

marginRight: 5,

},

open && { display: "none" },

]}

>

<MenuIcon />

</IconButton>

<Typography variant="h6" noWrap component="div">

School Management System [MULTIPLE]

</Typography>

</Toolbar>

</AppBar>

<Drawer variant="permanent" open={open}>

<DrawerHeader>

<IconButton onClick={handleDrawerClose}>

{theme.direction === "rtl" ? (

<ChevronRightIcon />

) : (

<ChevronLeftIcon />

)}

</IconButton>

</DrawerHeader>

<Divider />

<List>

{navArr.map((navItem, index) => (

<ListItem key={index} disablePadding sx={{ display: "block" }}>

<ListItemButton

sx={[

{

minHeight: 48,

px: 2.5,

},

open

? {

justifyContent: "initial",

}

: {

justifyContent: "center",

},

]}

onClick={() => {

handleNavigation(navItem.link);

}}

>

<ListItemIcon

sx={[

{

minWidth: 0,

justifyContent: "center",

},

open

? {

mr: 3,

}

: {

mr: "auto",

},

]}

>

{<navItem.icon />}

</ListItemIcon>

<ListItemText

primary={navItem.component}

sx={[

open

? {

opacity: 1,

}

: {

opacity: 0,

},

]}

/>

</ListItemButton>

</ListItem>

))}

</List>

<Divider />

{/\* <List>

{['All mail', 'Trash', 'Spam'].map((text, index) => (

<ListItem key={text} disablePadding sx={{ display: 'block' }}>

<ListItemButton

sx={[

{

minHeight: 48,

px: 2.5,

},

open

? {

justifyContent: 'initial',

}

: {

justifyContent: 'center',

},

]}

>

<ListItemIcon

sx={[

{

minWidth: 0,

justifyContent: 'center',

},

open

? {

mr: 3,

}

: {

mr: 'auto',

},

]}

>

{index % 2 === 0 ? <InboxIcon /> : <MailIcon />}

</ListItemIcon>

<ListItemText

primary={text}

sx={[

open

? {

opacity: 1,

}

: {

opacity: 0,

},

]}

/>

</ListItemButton>

</ListItem>

))}

</List> \*/}

</Drawer>

<Box component="main" sx={{ flexGrow: 1, p: 3 }}>

<DrawerHeader />

<Outlet />

</Box>

</Box>

);

}