You are an expert level professional level coder and programmer, Here you have to do is detect the error of student's is submitting the assignment and it say's assignment is submitted but the documents, files, video's are not being able to shown in the submitted section, in the provided code the assignments are being saved in backend folder named as uploads/assignments, but i want these to be shown in frontend folder and access in the website, there might me other issues too, detect the issue and provide me full updated error free code, here is the code of both backend and  teacher site dashboard frontend and student site dashboard frontend code,
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);
const Student = require('../models/student.model');
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: 'submissionVideo', maxCount: 1 },  // Add this for student videos
  { 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 - UPDATED
exports.createAssignment = [handleUpload, async (req, res) => {
  try {
    const {
      title,
```

```
  description,
  subject,
  class: classId,
  dueDate,
  videoUrl,
  rubric,
  maxPoints,
  allowLateSubmission,
  peerReviewEnabled
} = req.body;

// Convert to proper types
const assignment = new Assignment({
  title,
  description,
  subject,
  class: classId,
  dueDate: new Date(dueDate),
  rubric: rubric ? JSON.parse(rubric) : [],
  maxPoints: maxPoints ? Number(maxPoints) : 100,
  allowLateSubmission: allowLateSubmission === 'true' || allowLateSubmission === true,
  peerReviewEnabled: peerReviewEnabled === 'true' || peerReviewEnabled === true,
  school: req.user.schoolId,
  teacher: req.user.id
});

// Handle file uploads
if (req.files && req.files['attachments']) {
  req.files['attachments'].forEach(file => {
    assignment.attachments.push(`/uploads/assignments/${file.filename}`);
  });
}

// Handle video upload
if (req.files && req.files['assignmentVideo']) {
  const file = req.files['assignmentVideo'][0];
  assignment.videoUrl = `/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));
    } else if (videoUrl) {
      assignment.videoUrl = videoUrl;
    }

    await assignment.save();

    res.status(201).json({
      success: true,
      message: "Assignment created",
      data: assignment
    });
  } catch (error) {
    // Handle validation errors specifically
    if (error.name === 'ValidationError') {
      return res.status(400).json({
        success: false,
        message: "Validation Error",
        errors: Object.values(error.errors).map(e => e.message)
      });
    }

    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));
}


// Convert to proper types
const updateData = {
  title,
  description,
  subject,
  class: classId,
  dueDate: new Date(dueDate),
  videoUrl: assignmentVideoUrl,
  rubric: rubric ? JSON.parse(rubric) : [],
  maxPoints: maxPoints ? Number(maxPoints) : 100,
  allowLateSubmission: allowLateSubmission === 'true' || allowLateSubmission === true,
  peerReviewEnabled: peerReviewEnabled === 'true' || peerReviewEnabled === true,
  $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.findById(req.params.id);

    if (!assignment) {
      return res.status(404).json({
        success: false,
        message: "Assignment not found"
      });
    }

    // Optional: Delete associated files (attachments and video)
    const deleteFileIfExists = async (filePath) => {
      try {
        if (filePath && fs.existsSync(`.${filePath}`)) {
          await unlinkAsync(`.${filePath}`);
```

```
        }
      } catch (err) {
        console.warn(`Warning: Failed to delete file ${filePath}`, err.message);
      }
    };

    // Delete all attachments
    if (assignment.attachments && assignment.attachments.length > 0) {
      for (const file of assignment.attachments) {
        await deleteFileIfExists(file);
      }
    }

    // Delete assignment video if exists
    if (assignment.videoUrl) {
      await deleteFileIfExists(assignment.videoUrl);
    }

    // Finally delete the assignment
    await assignment.deleteOne();

    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 - Fixed version
exports.getAssignmentsForStudent = async (req, res) => {
  try {
    // Get student's class from their profile
    const student = await Student.findById(req.user.id).select('student_class');

    if (!student || !student.student_class) {
      return res.status(400).json({
```

```
        success: false,
        message: "Student class information not found. Please ensure you're assigned to a class."
      });
    }


    const assignments = await Assignment.find({
      class: student.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 // ✅ correct
);


// 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 { Controller, 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({
  defaultValues: {
    subject: "",
    class: "",
  },
});
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({
        subject: "",
        class: "",
      });
      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);

    // Ensure we have valid values for subject and class
    const subjectValue = subjects.some(
      (s) => s._id === assignment.subject?._id
    )
      ? assignment.subject?._id
      : "";

    const classValue = classes.some((c) => c._id === assignment.class?._id)
      ? assignment.class?._id
      : "";

    reset({
      title: assignment.title || "",
      description: assignment.description || "",
      subject: subjectValue,
```

```
      class: classValue,
      dueDate: formatDateForInput(assignment.dueDate) || "",
      videoUrl: assignment.videoUrl || "",
      maxPoints: assignment.maxPoints || 100,
      allowLateSubmission: assignment.allowLateSubmission || false,
      peerReviewEnabled: assignment.peerReviewEnabled || false,
      rubric: assignment.rubric || [],
    });
  },
  [reset, formatDateForInput, subjects, classes]
);

const handleCancelEdit = useCallback(() => {
  setIsEditing(false);
  setCurrentAssignment(null);
  reset({
    subject: "",
    class: "",
  });
}, [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 *"
    value={watch("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 *"
    value={watch("class") || ""}
    disabled={classes.length === 0}
  >
    <MenuItem value="">
      <em>Select Class</em>
    </MenuItem>
    {classes.length === 0 ? (
      <MenuItem disabled>Loading classes...</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: { 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 React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import {
 Container, Box, Typography, Card, CardContent, CardActions, Button,
 Chip, Avatar, Divider, LinearProgress, CircularProgress, IconButton,
 Dialog, DialogTitle, DialogContent, DialogActions, TextField,
 Snackbar, Alert, Badge, Tooltip, Grid, Paper, Stack, Tabs, Tab
} from '@mui/material';
import {
 Assignment, Description, CalendarToday, CheckCircle,
 Pending, Error, Upload, Download, Close,
 VideoLibrary, PlayCircle, Subject, Class, Person
} from '@mui/icons-material';
import { format, parseISO, isBefore } from 'date-fns';


const API_BASE = "http://localhost:5000/api";


const StudentAssignmentDashboard = () => {
 const [assignments, setAssignments] = useState([]);
 const [loading, setLoading] = useState(true);
```

```
const [error, setError] = useState(null);
const [selectedAssignment, setSelectedAssignment] = useState(null);
const [submissionDialogOpen, setSubmissionDialogOpen] = useState(false);
const [submissionFile, setSubmissionFile] = useState(null);
const [submissionVideo, setSubmissionVideo] = useState(null);
const [submissionRemarks, setSubmissionRemarks] = useState('');
const [videoUrl, setVideoUrl] = useState('');
const [submitting, setSubmitting] = useState(false);
const [successMessage, setSuccessMessage] = useState('');
const [submissions, setSubmissions] = useState([]);
const [activeTab, setActiveTab] = useState(0);
const navigate = useNavigate();


// Fetch assignments from the API
const fetchAssignments = async () => {
  try {
    setLoading(true);
    const token = localStorage.getItem('token');
    const response = await axios.get(`${API_BASE}/assignments/student/list`, {
      headers: { Authorization: `Bearer ${token}` }
    });
    setAssignments(response.data.data);
    setError(null);
  } catch (err) {
    setError('Failed to fetch assignments. Please try again later.');
    console.error('Error fetching assignments:', err);
  } finally {
    setLoading(false);
  }
};


// Fetch student submissions
const fetchSubmissions = async () => {
  try {
    const token = localStorage.getItem('token');
    const response = await axios.get(`${API_BASE}/assignments/submissions/student`, {
      headers: { Authorization: `Bearer ${token}` }
    });
    setSubmissions(response.data.data);
  } catch (err) {
    console.error('Error fetching submissions:', err);
  }
```

```
};


useEffect(() => {
  fetchAssignments();
  fetchSubmissions();
}, []);


const handleFileChange = (e) => {
  setSubmissionFile(e.target.files[0]);
};


const handleVideoChange = (e) => {
  setSubmissionVideo(e.target.files[0]);
};


const handleSubmitAssignment = async () => {
  if (!selectedAssignment) return;

  try {
    setSubmitting(true);
    const token = localStorage.getItem('token');
    const formData = new FormData();
    formData.append('assignmentId', selectedAssignment._id);
    formData.append('remarks', submissionRemarks);

    if (submissionFile) {
      formData.append('submission', submissionFile);
    }

    if (submissionVideo) {
      formData.append('feedbackVideo', submissionVideo);
    } else if (videoUrl) {
      formData.append('videoUrl', videoUrl);
    }

    const response = await axios.post(`${API_BASE}/assignments/submit`, formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
        Authorization: `Bearer ${token}`
      }
```

```
    });

    setSuccessMessage('Assignment submitted successfully!');
    fetchAssignments();
    fetchSubmissions();
    setSubmissionDialogOpen(false);
  } catch (err) {
    setError('Failed to submit assignment. Please try again.');
    console.error('Submission error:', err);
  } finally {
    setSubmitting(false);
  }
};


const handleDownloadAttachment = (url) => {
  window.open(`${API_BASE}${url}`, '_blank');
};


const handlePlayVideo = (url) => {
  if (url.startsWith('http')) {
    window.open(url, '_blank');
  } else {
    window.open(`${API_BASE}${url}`, '_blank');
  }
};


const getAssignmentStatus = (assignment, submission) => {
  const dueDate = parseISO(assignment.dueDate);
  const now = new Date();

  if (submission) {
    return {
      status: 'Submitted',
      color: 'success',
      icon: <CheckCircle />,
      text: `Submitted on ${format(parseISO(submission.createdAt), 'MMM dd, yyyy')}`,
      late: submission.lateSubmission
    };
  }

  if (isBefore(dueDate, now)) {
```

```
    return {
      status: 'Overdue',
      color: 'error',
      icon: <Error />,
      text: 'Past due date',
      late: true
    };
  }

  return {
    status: 'Pending',
    color: 'warning',
    icon: <Pending />,
    text: `Due ${format(dueDate, 'MMM dd, yyyy')}`,
    late: false
  };
};


const formatDate = (dateString) => {
  return format(parseISO(dateString), 'MMM dd, yyyy');
};


const getSubmissionStatus = (submission) => {
  if (submission.grade !== undefined) {
    return {
      status: 'Graded',
      color: 'success',
      text: `Grade: ${submission.grade}%`,
      icon: <CheckCircle />
    };
  }

  return {
    status: 'Submitted',
    color: 'info',
    text: 'Pending grading',
    icon: <Pending />
  };
};


return (
```

```
   <Container maxWidth="lg" sx={{ py: 4 }}>
     <Box sx={{ mb: 4 }}>
       <Typography variant="h4" component="h1" sx={{ fontWeight: 'bold', mb: 1 }}>
         <Assignment sx={{ verticalAlign: 'middle', mr: 2, fontSize: 40 }} />
         Assignment Dashboard
       </Typography>
       <Typography variant="body1" color="text.secondary">
         View your assignments, submit your work, and track your submissions
       </Typography>
     </Box>


     <Tabs value={activeTab} onChange={(e, newValue) => setActiveTab(newValue)} sx={{ mb: 3
}}>
       <Tab label="Active Assignments" />
       <Tab label="My Submissions" />
     </Tabs>


     {/* Active Assignments Tab */}
     {activeTab === 0 && (
       <>
         {loading ? (
           <Box sx={{ display: 'flex', justifyContent: 'center', mt: 4 }}>
             <CircularProgress size={60} />
           </Box>
         ) : error ? (
           <Alert severity="error" sx={{ mb: 3 }}>
             {error}
           </Alert>
         ) : assignments.length === 0 ? (
           <Card sx={{ p: 4, textAlign: 'center' }}>
             <Typography variant="h6" color="text.secondary">
               No assignments available at this time
             </Typography>
             <Typography variant="body2" color="text.secondary" sx={{ mt: 1 }}>
               Check back later or contact your teacher
             </Typography>
           </Card>
         ) : (
           <Grid container spacing={3}>
             {assignments.map((assignment) => {
               const submission = submissions.find(s => s.assignment === assignment._id);
               const status = getAssignmentStatus(assignment, submission);
```

```
return (
  <Grid item xs={12} md={6} key={assignment._id}>
    <Card
      sx={{
        height: '100%',
        display: 'flex',
        flexDirection: 'column',
        borderLeft: `4px solid ${
          status.color === 'success' ? '#4caf50' :
          status.color === 'warning' ? '#ff9800' :
          '#f44336'
        }`
      }}
    >
      <CardContent sx={{ flexGrow: 1 }}>
        <Box sx={{ display: 'flex', justifyContent: 'space-between', mb: 1 }}>
          <Chip
            label={assignment.subject.subject_name}
            size="small"
            sx={{
              bgcolor: '#e3f2fd',
              color: '#1976d2',
              fontWeight: 'bold'
            }}
          />
          <Chip
            icon={status.icon}
            label={status.status}
            color={status.color}
            size="small"
            variant="outlined"
          />
        </Box>

        <Typography variant="h6" sx={{ mt: 1, fontWeight: 'bold' }}>
          {assignment.title}
        </Typography>

        <Typography variant="body2" color="text.secondary" sx={{ mt: 1, mb: 2 }}>
          {assignment.description}
        </Typography>

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

```
<Grid container spacing={1} sx={{ mb: 2 }}>
  <Grid item xs={6}>
    <Box sx={{ display: 'flex', alignItems: 'center' }}>
      <Class sx={{ fontSize: 16, color: 'text.secondary', mr: 1 }} />
      <Typography variant="body2">
        Class: {assignment.class.class_text}
      </Typography>
    </Box>
  </Grid>
  <Grid item xs={6}>
    <Box sx={{ display: 'flex', alignItems: 'center' }}>
      <Person sx={{ fontSize: 16, color: 'text.secondary', mr: 1 }} />
      <Typography variant="body2">
        Teacher: {assignment.teacher.name}
      </Typography>
    </Box>
  </Grid>
  <Grid item xs={12}>
    <Box sx={{ display: 'flex', alignItems: 'center' }}>
      <CalendarToday sx={{ fontSize: 16, color: 'text.secondary', mr: 1 }} />
      <Typography variant="body2" color={status.late ? 'error' : 'inherit'}>
        {status.text}
      </Typography>
      {status.late && (
        <Chip
          label="Late"
          size="small"
          color="error"
          sx={{ ml: 1, fontSize: '0.7rem' }}
        />
      )}
    </Box>
  </Grid>
</Grid>

{assignment.attachments.length > 0 && (
  <Box sx={{ mt: 1 }}>
    <Typography variant="body2" sx={{ fontWeight: 'bold', mb: 1 }}>
      Attachments:
    </Typography>
    <Stack spacing={1}>
      {assignment.attachments.map((file, index) => (
        <Chip
```

```
            key={index}
            icon={<Description />}
            label={file.split('/').pop()}
            onClick={() => handleDownloadAttachment(file)}
            sx={{
              cursor: 'pointer',
              justifyContent: 'flex-start',
              maxWidth: '100%',
              overflow: 'hidden',
              textOverflow: 'ellipsis'
            }}
          />
        ))}
      </Stack>
    </Box>
  )}

  {assignment.videoUrl && (
    <Box sx={{ mt: 2 }}>
      <Typography variant="body2" sx={{ fontWeight: 'bold', mb: 1 }}>
        Instructional Video:
      </Typography>
      <Chip
        icon={<VideoLibrary />}
        label="Watch Video"
        onClick={() => handlePlayVideo(assignment.videoUrl)}
        sx={{ cursor: 'pointer' }}
        color="primary"
      />
    </Box>
  )}

  <Box sx={{ mt: 2, display: 'flex', justifyContent: 'space-between' }}>
    <Typography variant="body2" sx={{ fontWeight: 'bold' }}>
      Max Points: {assignment.maxPoints}
    </Typography>
    {assignment.peerReviewEnabled && (
      <Chip label="Peer Review" size="small" color="secondary" />
    )}
  </Box>
</CardContent>

<CardActions sx={{ justifyContent: 'flex-end', p: 2 }}>
  {!submission ? (
```

```
                    <Button
                      variant="contained"
                      startIcon={<Upload />}
                      onClick={() => {
                        setSelectedAssignment(assignment);
                        setSubmissionDialogOpen(true);
                      }}
                      disabled={isBefore(parseISO(assignment.dueDate), new Date()) &&
!assignment.allowLateSubmission}
                    >
                      {isBefore(parseISO(assignment.dueDate), new Date()) &&
!assignment.allowLateSubmission
                        ? 'Submission Closed'
                        : 'Submit Assignment'}
                    </Button>
                  ) : (
                    <Button
                      variant="outlined"
                      onClick={() => {
                        const submission = submissions.find(s => s.assignment ===
assignment._id);
                        if (submission) {
                          setActiveTab(1);
                          setTimeout(() => {

document.getElementById(`submission-${submission._id}`)?.scrollIntoView({
                              behavior: 'smooth'
                            });
                          }, 100);
                        }
                      }}
                    >
                      View Submission
                    </Button>
                  )}
                </CardActions>
              </Card>
            </Grid>
          );
        })}
      </Grid>
    )}
  </>
)}
```

```
{/* My Submissions Tab */}
{activeTab === 1 && (
  <Box>
    {submissions.length === 0 ? (
      <Card sx={{ p: 4, textAlign: 'center' }}>
        <Typography variant="h6" color="text.secondary">
          You haven't submitted any assignments yet
        </Typography>
        <Typography variant="body2" color="text.secondary" sx={{ mt: 1 }}>
          Submit your first assignment to see it here
        </Typography>
      </Card>
    ) : (
      <Grid container spacing={3}>
        {submissions.map((submission) => {
          const assignment = assignments.find(a => a._id === submission.assignment);
          const status = getSubmissionStatus(submission);

          if (!assignment) return null;

          return (
            <Grid item xs={12} key={submission._id} id={`submission-${submission._id}`}>
              <Paper elevation={2} sx={{ p: 3 }}>
                <Box sx={{ display: 'flex', justifyContent: 'space-between', mb: 2 }}>
                  <Typography variant="h6" sx={{ fontWeight: 'bold' }}>
                    {assignment.title}
                  </Typography>
                  <Chip
                    icon={status.icon}
                    label={status.status}
                    color={status.color}
                    size="small"
                  />
                </Box>

                <Grid container spacing={2}>
                  <Grid item xs={12} md={6}>
                    <Typography variant="body2" color="text.secondary">
                      Submitted on: {formatDate(submission.createdAt)}
                    </Typography>

                    {submission.lateSubmission && (
```

```
  <Chip
    label="Late Submission"
    size="small"
    color="error"
    sx={{ mt: 1 }}
  />
)}

{submission.remarks && (
  <Box sx={{ mt: 2 }}>
    <Typography variant="body2" sx={{ fontWeight: 'bold' }}>
      Your Remarks:
    </Typography>
    <Typography variant="body2" sx={{ mt: 1 }}>
      {submission.remarks}
    </Typography>
  </Box>
)}

{submission.fileUrl && (
  <Box sx={{ mt: 2 }}>
    <Typography variant="body2" sx={{ fontWeight: 'bold', mb: 1 }}>
      Submitted File:
    </Typography>
    <Button
      variant="outlined"
      startIcon={<Download />}
      onClick={() => handleDownloadAttachment(submission.fileUrl)}
    >
      Download File
    </Button>
  </Box>
)}

{(submission.videoUrl || submission.feedbackVideoUrl) && (
  <Box sx={{ mt: 2 }}>
    <Typography variant="body2" sx={{ fontWeight: 'bold', mb: 1 }}>
      {submission.videoUrl ? 'Submitted Video:' : 'Feedback Video:'}
    </Typography>
    <Button
      variant="outlined"
      startIcon={<PlayCircle />}
      onClick={() =>
        handlePlayVideo(submission.videoUrl || submission.feedbackVideoUrl)
```

```
      }
    >
      {submission.videoUrl ? 'Watch Your Video' : 'Watch Feedback'}
    </Button>
  </Box>
 )}
</Grid>

<Grid item xs={12} md={6}>
 {submission.grade !== undefined && (
  <Box sx={{ mb: 3 }}>
   <Typography variant="body2" sx={{ fontWeight: 'bold', mb: 1 }}>
    Grade: {submission.grade}%
   </Typography>
   <LinearProgress
    variant="determinate"
    value={submission.grade}
    sx={{ height: 10, borderRadius: 5 }}
    color={
     submission.grade >= 90 ? 'success' :
     submission.grade >= 70 ? 'primary' :
     'error'
    }
   />
  </Box>
 )}

 {submission.feedback && (
  <Box sx={{
   bgcolor: '#f5f5f5',
   p: 2,
   borderRadius: 1,
   borderLeft: '3px solid #3f51b5'
  }}>
   <Typography variant="body2" sx={{ fontWeight: 'bold', mb: 1 }}>
    Teacher Feedback:
   </Typography>
   <Typography variant="body2">
    {submission.feedback}
   </Typography>
  </Box>
 )}

 {assignment.rubric.length > 0 && submission.rubricScores && (
```

```
                  <Box sx={{ mt: 3 }}>
                    <Typography variant="body2" sx={{ fontWeight: 'bold', mb: 1 }}>
                      Rubric Assessment:
                    </Typography>
                    {assignment.rubric.map((criteria, index) => {
                      const score = submission.rubricScores.find(
                        s => s.criteriaId === criteria._id
                      )?.score;

                      return (
                        <Box key={index} sx={{ mb: 1 }}>
                          <Typography variant="body2">
                            {criteria.name}: {score || 0}/{criteria.maxScore}
                          </Typography>
                          <LinearProgress
                            variant="determinate"
                            value={(score || 0) / criteria.maxScore * 100}
                            sx={{ height: 6, borderRadius: 3 }}
                          />
                        </Box>
                      );
                    })}
                  </Box>
                )}
              </Grid>
            </Grid>
          </Paper>
        </Grid>
      );
    })}
  </Grid>
 )}
</Box>
)}


{/* Submission Dialog */}
<Dialog
  open={submissionDialogOpen}
  onClose={() => setSubmissionDialogOpen(false)}
  maxWidth="md"
  fullWidth
>
  <DialogTitle>
```

```
          <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
            <Typography variant="h6">
              Submit Assignment: {selectedAssignment?.title}
            </Typography>
            <IconButton onClick={() => setSubmissionDialogOpen(false)}>
              <Close />
            </IconButton>
          </Box>
        </DialogTitle>

        <DialogContent>
          {selectedAssignment && (
            <>
              <Typography variant="body1" sx={{ mb: 2 }}>
                {selectedAssignment.description}
              </Typography>

              <Box sx={{ mb: 3, p: 2, bgcolor: '#f9f9f9', borderRadius: 1 }}>
                <Grid container spacing={1}>
                  <Grid item xs={12} sm={6}>
                    <Typography variant="body2">
                      <strong>Subject:</strong> {selectedAssignment.subject.subject_name}
                    </Typography>
                  </Grid>
                  <Grid item xs={12} sm={6}>
                    <Typography variant="body2">
                      <strong>Class:</strong> {selectedAssignment.class.class_text}
                    </Typography>
                  </Grid>
                  <Grid item xs={12} sm={6}>
                    <Typography variant="body2">
                      <strong>Teacher:</strong> {selectedAssignment.teacher.name}
                    </Typography>
                  </Grid>
                  <Grid item xs={12} sm={6}>
                    <Typography variant="body2"
color={isBefore(parseISO(selectedAssignment.dueDate), new Date()) ? 'error' : 'inherit'}>
                      <strong>Due Date:</strong> {formatDate(selectedAssignment.dueDate)}
                    </Typography>
                  </Grid>
                  <Grid item xs={12} sm={6}>
                    <Typography variant="body2">
                      <strong>Max Points:</strong> {selectedAssignment.maxPoints}
                    </Typography>
```

```
    </Grid>
  </Grid>
</Box>

<TextField
  label="Remarks (optional)"
  multiline
  rows={3}
  fullWidth
  value={submissionRemarks}
  onChange={(e) => setSubmissionRemarks(e.target.value)}
  sx={{ mb: 3 }}
/>

<Box sx={{ mb: 3 }}>
  <Typography variant="body1" sx={{ fontWeight: 'bold', mb: 1 }}>
    Upload Assignment File (PDF, DOC, etc.)
  </Typography>
  <Button
    variant="outlined"
    component="label"
    startIcon={<Upload />}
    sx={{ mr: 2 }}
  >
    Choose File
    <input
      type="file"
      hidden
      onChange={handleFileChange}
      accept=".pdf,.doc,.docx,.txt,.jpg,.jpeg,.png"
    />
  </Button>
  {submissionFile && (
    <Typography variant="body2" sx={{ mt: 1 }}>
      Selected: {submissionFile.name}
    </Typography>
  )}
</Box>

<Box sx={{ mb: 3 }}>
  <Typography variant="body1" sx={{ fontWeight: 'bold', mb: 1 }}>
    Video Submission
  </Typography>
```

```
  <Box sx={{ display: 'flex', flexDirection: 'column', gap: 2 }}>
    <Typography variant="body2">
      Option 1: Upload a video file
    </Typography>
    <Button
      variant="outlined"
      component="label"
      startIcon={<Upload />}
      sx={{ alignSelf: 'flex-start' }}
    >
      Choose Video File
      <input
        type="file"
        hidden
        onChange={handleVideoChange}
        accept="video/*"
      />
    </Button>
    {submissionVideo && (
      <Typography variant="body2">
        Selected: {submissionVideo.name}
      </Typography>
    )}

    <Typography variant="body2" sx={{ mt: 2 }}>
      Option 2: Provide a video URL (YouTube, Vimeo, etc.)
    </Typography>
    <TextField
      label="Video URL"
      fullWidth
      value={videoUrl}
      onChange={(e) => setVideoUrl(e.target.value)}
      placeholder="https://www.youtube.com/watch?v=..."
    />
  </Box>
</Box>

{selectedAssignment.attachments.length > 0 && (
  <Box sx={{ mb: 3 }}>
    <Typography variant="body1" sx={{ fontWeight: 'bold', mb: 1 }}>
      Assignment Attachments
    </Typography>
    <Stack spacing={1}>
      {selectedAssignment.attachments.map((file, index) => (
```

```
                <Chip
                  key={index}
                  icon={<Description />}
                  label={file.split('/').pop()}
                  onClick={() => handleDownloadAttachment(file)}
                  sx={{ cursor: 'pointer', justifyContent: 'flex-start' }}
                />
              ))}
            </Stack>
          </Box>
        )}
      </>
    )}
  </DialogContent>

  <DialogActions sx={{ p: 3 }}>
    <Button onClick={() => setSubmissionDialogOpen(false)}>
      Cancel
    </Button>
    <Button
      variant="contained"
      onClick={handleSubmitAssignment}
      disabled={submitting || (!submissionFile && !submissionVideo && !videoUrl)}
      startIcon={submitting ? <CircularProgress size={20} /> : <Upload />}
    >
      {submitting ? 'Submitting...' : 'Submit Assignment'}
    </Button>
  </DialogActions>
</Dialog>


{/* Success Snackbar */}
<Snackbar
  open={!!successMessage}
  autoHideDuration={5000}
  onClose={() => setSuccessMessage('')}
  anchorOrigin={{ vertical: 'top', horizontal: 'right' }}
>
  <Alert onClose={() => setSuccessMessage('')} severity="success" sx={{ width: '100%' }}>
    {successMessage}
  </Alert>
</Snackbar>
</Container>
);
```

};

export default StudentAssignmentDashboard;
Provide me full updated code