You are an expert full stack developer, study this given code, create me an frontend for assignment for teacher's adding the assignment and viewing the assignment submitted by students, i do have code from of backend for assignment, i do have full code of student, teacher, class , here is the full code in one file of frontend:

```
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');
const cookieParser = require('cookie-parser');
const path = require('path');
const dotenv = require('dotenv');
const schoolRouter = require('./routers/school.router.js');
const classRouter = require("./routers/class.router.js");
const subjectRouter = require("./routers/subject.router.js")
const studentRouter = require('./routers/student.router.js');
const teacherRouter = require('./routers/teacher.router.js');
const scheduleRouter = require('./routers/schedule.router.js')
const attendanceRouter = require('./routers/attendance.router.js');
const examinationRouter = require("./routers/examination.router.js");
const noticeRouter = require("./routers/notice.router.js");
const teacherAttendanceRouter = require("./routers/teacherAttendance.router.js")
const chatbotRoutes = require("./routers/chatbot.router.js");
const assignmentRoutes = require('./routers/assignment.router.js');




dotenv.config();


const app = express();


// 1. Enhanced CORS configuration


app.use(cors({
  origin: 'http://localhost:5173',
  methods: ['GET', 'POST', 'PUT', 'DELETE','PATCH','OPTIONS'],
  allowedHeaders: ['Content-Type', 'Authorization'],
  credentials: true,
  exposedHeaders: ['Authorization']
}));
```

```
app.use('/images/uploaded/school', express.static(
  path.join(__dirname, '../frontend/public/images/uploaded/school'),
  { maxAge: '1d' }
));


// 2. Middleware ordering fix
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());




// 4. Improved MongoDB connection
mongoose.connect(process.env.MONGODB_URI)
  .then(() => console.log('MongoDB connected successfully'))
  .catch(err => console.error('MongoDB connection error:', err));


// 5. Health check endpoint
app.get('/health', (req, res) => {
  res.status(200).json({
    status: 'OK',
    timestamp: new Date().toISOString()
  });
});


// 6. API routes
app.use('/api/school', schoolRouter);
app.use("/api/class",classRouter);
app.use("/api/subjects", subjectRouter);
app.use('/api/students', studentRouter);
app.use("/api/teachers",teacherRouter)
app.use("/api/schedule",scheduleRouter);
app.use("/api/attendance",attendanceRouter);
app.use("/api/examination",examinationRouter);
app.use("/api/notice",noticeRouter);
app.use("/api/teacherAttendance",teacherAttendanceRouter);
app.use("/api/chatbot", chatbotRoutes);
app.use('/api/assignments', assignmentRoutes);
```

```
// 7. Enhanced error handling
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    success: false,
    message: 'Internal Server Error',
    error: process.env.NODE_ENV === 'production' ? undefined : err.message
  });
});


const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
  console.log(`Environment: ${process.env.NODE_ENV || 'development'}`);
});

const express = require('express');
const router = express.Router();
const teacherController = require('../controllers/teacher.controller');
const authMiddleware = require('../auth/auth');
const authController = require('../controllers/authController');


// Public routes
router.post('/register', authMiddleware(['SCHOOL']), teacherController.registerTeacher);
router.post('/login', teacherController.loginTeacher);
router.get('/validate-reset-token/:token', authController.validateResetToken);
router.post('/forgot-password', authController.forgotPassword);
router.post('/reset-password/:token', authController.resetPassword);


// Protected routes
router.get('/all', authMiddleware(['SCHOOL','TEACHER']), teacherController.getAllTeachers);
router.get('/fetch-single', authMiddleware(['TEACHER','SCHOOL']),
teacherController.getTeacherOwnData);
router.get('/class-teacher/:classId', authMiddleware(['SCHOOL', 'TEACHER']),
teacherController.getClassTeacher);
```

```
router.get('/:id', authMiddleware(['STUDENT','SCHOOL','TEACHER']),
teacherController.getTeacherById);
router.patch('/update', authMiddleware(['TEACHER', 'SCHOOL']),
teacherController.updateTeacher);
router.delete('/delete/:id', authMiddleware(['SCHOOL']), teacherController.deleteTeacherWithId);






module.exports = router;


const mongoose = require('mongoose');


const teacherSchema = new mongoose.Schema({
    school: { type: mongoose.Schema.Types.ObjectId, ref: 'School' },
    email: { type: String, required: true, unique: true },
    name: { type: String, required: true },
    qualification: { type: String, required: true },
    age: { type: Number, required: true },
    gender: { type: String, required: true, enum: ['Male', 'Female', 'Other'] },
    teacher_image: { type: String, required: true },
    password: { type: String, required: true },
    class: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Class',
        default: null
    },
    is_class_teacher: {
        type: Boolean,
        default: false
    },
    subjects: [{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Subject'
    }],
   resetPasswordToken: {
    type: String, // ✅ FIXED: Should be string, not Object
  },
  resetPasswordExpires: {
    type: Date, // ✅ FIXED: Should be Date, not Object
```

```
 },
  createdAt: { type: Date, default: Date.now }
});


// Pre-save hook to ensure only one class teacher per class
teacherSchema.pre('save', async function(next) {
  if (this.isModified('is_class_teacher') && this.is_class_teacher && this.class) {
    const existingClassTeacher = await mongoose.model('Teacher').findOne({
      class: this.class,
      is_class_teacher: true,
      _id: { $ne: this._id }
    });

    if (existingClassTeacher) {
      throw new Error(`Class already has a class teacher: ${existingClassTeacher.name}`);
    }
  }
  next();
});


module.exports = mongoose.model('Teacher', teacherSchema);

const formidable = require('formidable');
const Teacher = require('../models/teacher.model');
const path = require('path');
const fs = require('fs');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const mongoose = require('mongoose');


// Helper function for file uploads
const handleFileUpload = (files, uploadPath, fieldName) => {
  if (!files?.[fieldName]) {
    throw new Error(`${fieldName} is required`);
  }


  const photo = files[fieldName][0];
  const sanitizedFilename = `${Date.now()}-${photo.originalFilename
    .replace(/ /g, "_")
    .replace(/[^a-zA-Z0-9_.-]/g, "")
```

```
        .toLowerCase()}`;


    const newPath = path.join(uploadPath, sanitizedFilename);
    fs.renameSync(photo.filepath, newPath);


    return sanitizedFilename;
};


// Register a new teacher
exports.registerTeacher = async (req, res) => {
    try {
        const form = new formidable.IncomingForm();


        form.parse(req, async (err, fields, files) => {
            try {
                // Validate input fields
                const requiredFields = ['name', 'email', 'qualification', 'age', 'gender', 'password'];
                const missingFields = requiredFields.filter(field => !fields[field]);


                if (missingFields.length > 0) {
                    return res.status(400).json({
                        success: false,
                        message: `Missing required fields: ${missingFields.join(', ')}`
                    });
                }


                const email = fields.email[0].trim().toLowerCase();
                const rawPassword = fields.password[0].trim();
                const classId = fields.class?.[0] || null;
                const isClassTeacher = fields.is_class_teacher?.[0] === 'true';


                // Email validation
                if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {
                    return res.status(400).json({
                        success: false,
                        message: "Invalid email format"
                    });
                }


                // Password validation
                if (rawPassword.length < 8) {
                    return res.status(400).json({
                        success: false,
                        message: "Password must be at least 8 characters"
                    });
                }


                // Check if teacher already exists
                const existingTeacher = await Teacher.findOne({
                    email: { $regex: new RegExp(`^${email}$`, "i") }
                });


                if (existingTeacher) {
                    return res.status(409).json({
                        success: false,
                        message: "Email already registered"
                    });
                }


                // Check if class teacher assignment is valid
                if (isClassTeacher && classId) {
                    const existingClassTeacher = await Teacher.findOne({
                        class: classId,
                        is_class_teacher: true
                    });

                    if (existingClassTeacher) {
                        return res.status(400).json({
                            success: false,
                            message: `Class already has a class teacher: ${existingClassTeacher.name}`
                        });
                    }
                }


                // Handle image upload
                const uploadDir = path.join(process.cwd(), process.env.TEACHER_IMAGE_PATH);
```

```
    if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir, { recursive: true, mode: 0o755 });
    }


    const teacherImage = handleFileUpload(files, uploadDir, 'teacher_image');


    // Create new teacher
    const hashedPassword = bcrypt.hashSync(rawPassword, 10);


    const newTeacher = new Teacher({
      school: req.user.schoolId,
      email: email,
      name: fields.name[0],
      qualification: fields.qualification[0],
      age: fields.age[0],
      gender: fields.gender[0],
      teacher_image: teacherImage,
      password: hashedPassword,
      class: classId,
      is_class_teacher: isClassTeacher,
      subjects: fields.subjects?.[0]?.split(',') || []
    });


    const savedTeacher = await newTeacher.save();


    res.status(201).json({
      success: true,
      data: {
        id: savedTeacher._id,
        email: savedTeacher.email,
        name: savedTeacher.name,
        is_class_teacher: savedTeacher.is_class_teacher
      },
      message: "Teacher registered successfully"
    });


  } catch (error) {
    console.error("Teacher registration error:", error);
```

```
            res.status(500).json({
                success: false,
                message: "Teacher registration failed",
                error: process.env.NODE_ENV === 'development' ? error.message : undefined
            });
        }
    });
} catch (error) {
    res.status(500).json({
        success: false,
        message: "Server error",
        error: process.env.NODE_ENV === 'development' ? error.message : undefined
    });
}
};


// Login teacher
exports.loginTeacher = async (req, res) => {
    try {
        const { email, password } = req.body;


        if (!email || !password) {
            return res.status(400).json({
                success: false,
                message: "Email and password are required"
            });
        }


        const teacher = await Teacher.findOne({
            email: { $regex: new RegExp(`^${email.trim()}$`, 'i') }
        }).populate('school class subjects');


        if (!teacher) {
            return res.status(401).json({
                success: false,
                message: "Invalid credentials"
            });
        }
        const isMatch = await bcrypt.compare(password.trim(), teacher.password);
        if (!isMatch) {
            return res.status(401).json({
                success: false,
                message: "Invalid credentials"
            });
        }


        const payload = {
            id: teacher._id.toString(),
            schoolId: teacher.school?._id.toString(),
            name: teacher.name,
            image_url: teacher.teacher_image,
            role: "TEACHER",
            email: teacher.email
        };


        const token = jwt.sign(
            payload,
            process.env.TEACHER_JWT_SECRET,
            { expiresIn: '7d', algorithm: 'HS256' }
        );


        res.status(200).json({
            success: true,
            message: "Login successful",
            token: token,
            user: payload
        });


    } catch (error) {
        console.error("Teacher login error:", error);
        res.status(500).json({
            success: false,
            message: "Login failed",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};
```

```
// Get all teachers
exports.getAllTeachers = async (req, res) => {
    try {
        const filter = {};
        if (req.user.role === 'SCHOOL') {
            filter.school = req.user.schoolId;
        }


        const teachers = await Teacher.find(filter)
            .select('-password -__v')
            .populate('class', 'name class_text')
            .populate('subjects', 'name')
            .lean();


        res.status(200).json({
            success: true,
            message: "Teachers fetched successfully",
            count: teachers.length,
            data: teachers
        });
    } catch (error) {
        console.error("Get all teachers error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to fetch teachers",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Get teacher by ID
exports.getTeacherById = async (req, res) => {
    try {
        if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
            return res.status(400).json({
                success: false,
                message: "Invalid teacher ID format"
            });
        }
        const filter = { _id: req.params.id };
        if (req.user.role === 'SCHOOL') {
            filter.school = req.user.schoolId;
        }


        const teacher = await Teacher.findOne(filter)
            .select('-password')
            .populate('class', 'name class_text')
            .populate('subjects', 'name')
            .lean();


        if (!teacher) {
            return res.status(404).json({
                success: false,
                message: "Teacher not found or unauthorized"
            });
        }


        res.status(200).json({
            success: true,
            data: teacher
        });


    } catch (error) {
        console.error("Get teacher error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to fetch teacher data",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Update teacher
exports.updateTeacher = async (req, res) => {
    try {
        const form = new formidable.IncomingForm();
```

```
form.parse(req, async (err, fields, files) => {
  try {
    let teacher;
    if (req.user.role === 'TEACHER') {
      teacher = await Teacher.findById(req.user.id);
    } else if (req.user.role === 'SCHOOL') {
      const teacherId = fields.teacherId?.[0] || req.user.id;
      teacher = await Teacher.findOne({
        _id: teacherId,
        school: req.user.schoolId
      });
    } else {
      return res.status(403).json({
        success: false,
        message: "Unauthorized access"
      });
    }


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


    // Handle class teacher assignment
    const newClassId = fields.class?.[0] || null;
    const newIsClassTeacher = fields.is_class_teacher?.[0] === 'true';

    if (newIsClassTeacher && newClassId) {
      // Check if another teacher is already class teacher of this class
      const existingClassTeacher = await Teacher.findOne({
        class: newClassId,
        is_class_teacher: true,
        _id: { $ne: teacher._id }
      });

      if (existingClassTeacher) {
        return res.status(400).json({
          success: false,
          message: `Class already has a class teacher: ${existingClassTeacher.name}`
        });
    }
}


// Handle image update
if (files?.teacher_image) {
    const uploadDir = path.join(process.cwd(), process.env.TEACHER_IMAGE_PATH);

    // Delete old image
    if (teacher.teacher_image) {
        const oldPath = path.join(uploadDir, teacher.teacher_image);
        if (fs.existsSync(oldPath)) {
            fs.unlinkSync(oldPath);
        }
    }


    const teacherImage = handleFileUpload(files, uploadDir, 'teacher_image');
    teacher.teacher_image = teacherImage;
}


// Update fields
const allowedUpdates = [
    'name', 'qualification', 'age', 'gender',
    'class', 'is_class_teacher', 'subjects'
];

allowedUpdates.forEach(field => {
    if (fields[field]) {
        if (field === 'subjects') {
            teacher[field] = fields[field][0]?.split(',') || [];
        } else if (field === 'class') {
            teacher[field] = fields[field][0] || null;
        } else if (field === 'is_class_teacher') {
            teacher[field] = fields[field][0] === 'true';
        } else {
            teacher[field] = fields[field][0].trim();
        }
    }
});
```

```
            // Handle password update
            if (fields.password) {
                const newPassword = fields.password[0].trim();
                if (newPassword.length < 8) {
                    return res.status(400).json({
                        success: false,
                        message: "Password must be at least 8 characters"
                    });
                }
                teacher.password = bcrypt.hashSync(newPassword, 10);
            }


            await teacher.save();


            const updatedTeacher = await Teacher.findById(teacher._id)
                .select('-password -__v')
                .populate('class', 'name class_text')
                .populate('subjects', 'name')
                .lean();


            res.status(200).json({
                success: true,
                message: "Teacher updated successfully",
                data: updatedTeacher
            });


        } catch (error) {
            console.error("Teacher update error:", error);
            res.status(500).json({
                success: false,
                message: "Failed to update teacher",
                error: process.env.NODE_ENV === 'development' ? error.message : undefined
            });
        }
    });
} catch (error) {
    console.error("Update teacher error:", error);
    res.status(500).json({
        success: false,
        message: "Internal server error",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Get class teacher for a class
exports.getClassTeacher = async (req, res) => {
    try {
        const classId = req.params.classId;

        if (!mongoose.Types.ObjectId.isValid(classId)) {
            return res.status(400).json({
                success: false,
                message: "Invalid class ID format"
            });
        }


        const classTeacher = await Teacher.findOne({
            class: classId,
            is_class_teacher: true,
            school: req.user.schoolId
        })
        .select('_id name email teacher_image')
        .lean();


        if (!classTeacher) {
            return res.status(404).json({
                success: false,
                message: "No class teacher assigned for this class"
            });
        }


        res.status(200).json({
            success: true,
            data: classTeacher
        });


    } catch (error) {
        console.error("Get class teacher error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to fetch class teacher",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Delete teacher
exports.deleteTeacherWithId = async (req, res) => {
    try {
        if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
            return res.status(400).json({
                success: false,
                message: "Invalid teacher ID format"
            });
        }


        const teacher = await Teacher.findOneAndDelete({
            _id: req.params.id,
            school: req.user.schoolId
        });


        if (!teacher) {
            return res.status(404).json({
                success: false,
                message: "Teacher not found or not authorized to delete"
            });
        }


        // Delete associated image
        if (teacher.teacher_image) {
            const imagePath = path.join(
                process.cwd(),
                process.env.TEACHER_IMAGE_PATH,
                teacher.teacher_image
            );

            if (fs.existsSync(imagePath)) {
                fs.unlinkSync(imagePath);
            }
        }


        res.status(200).json({
            success: true,
            message: "Teacher deleted successfully"
        });


    } catch (error) {
        console.error("Delete teacher error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to delete teacher",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Get teacher's own data
exports.getTeacherOwnData = async (req, res) => {
    try {
        const teacher = await Teacher.findById(req.user.id)
            .select('-password')
            .populate('class', 'name class_text')
            .populate('subjects', 'name')
            .lean();


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


        res.status(200).json({
            success: true,
            data: teacher
        });
    } catch (error) {
        console.error("Get teacher data error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to fetch teacher data",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};

const express = require('express');
const router = express.Router();
const studentController = require('../controllers/student.controller');
const authMiddleware = require('../auth/auth');
const authController = require('../controllers/authController');




// Public routes
router.post('/register', authMiddleware(['SCHOOL']), studentController.registerStudent);
router.post('/login', studentController.loginStudent);
router.get('/validate-reset-token/:token', authController.validateResetToken);
router.post('/forgot-password', authController.forgotPassword);
router.post('/reset-password/:token', authController.resetPassword);
// Protected routes
router.get('/all', authMiddleware(['SCHOOL','TEACHER']),
studentController.getStudentsWithQuery);
router.get('/fetch-single', authMiddleware(['STUDENT','SCHOOL','TEACHER']),
studentController.getStudentOwnData);
router.get('/:id', authMiddleware(['SCHOOL','TEACHER']), studentController.getStudentById);
router.patch('/update', authMiddleware(['STUDENT', 'SCHOOL']),
studentController.updateStudent);
router.delete('/delete/:id', authMiddleware(['SCHOOL']), studentController.deleteStudentWithId);


// Additional route
router.get('/', authMiddleware(['SCHOOL','TEACHER','STUDENT']),
studentController.getAllStudents);
```

```
module.exports = router;

const mongoose = require('mongoose');


// Define the schema for Student
const studentSchema = new mongoose.Schema(
  {
    school: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'School', // Reference to the 'School' model
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true, // Ensures unique email addresses
      trim: true, // Removes extra spaces
      lowercase: true, // Converts to lowercase
    },
    name: {
      type: String,
      required: true,
      trim: true,
    },
    student_class: {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'Class',
      validate: {
        validator: mongoose.Types.ObjectId.isValid,
        message: props => `${props.value} is not a valid class id!`
      },
      default: null
    },
    age: {
      type: String,
      required: true,
    },
    gender: {
      type: String,
      required: true,
      enum: ['Male', 'Female', 'Other'], // Restricts values
    },
    guardian: {
      type: String,
      required: true,
    },
    guardian_phone: {
      type: String,
      required: true,
      match: [/^\d{10}$/, 'Invalid phone number'], // Ensures a 10-digit number
    },
    student_image: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
      minlength: 6, // Ensures a minimum password length
    },
    createdAt: {
      type: Date,
      default: Date.now, // Uses the current timestamp
    },
    resetPasswordToken: {
    type: String, // ✅ FIXED: Should be string, not Object
  },
  resetPasswordExpires: {
    type: Date, // ✅ FIXED: Should be Date, not Object
  },
  },
  { timestamps: true } // Enables 'createdAt' and 'updatedAt'
);


// Export the model
module.exports = mongoose.model('Student', studentSchema);





require("dotenv").config();
const formidable = require("formidable");
const Student = require("../models/student.model");
const path = require("path");
const fs = require("fs");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const mongoose = require('mongoose');


// Helper function for file upload
const handleFileUpload = (files, uploadPath, fieldName) => {
    if (!files?.[fieldName]) {
      throw new Error(`${fieldName} is required`);
    }


    const photo = files[fieldName][0];
    const sanitizedFilename = `${Date.now()}-${photo.originalFilename
      .replace(/ /g, "_")
      .replace(/[^a-zA-Z0-9_.-]/g, "")
      .toLowerCase()}`;


    const newPath = path.join(uploadPath, sanitizedFilename);
    fs.renameSync(photo.filepath, newPath);


    return sanitizedFilename;
};


// Register a new student
exports.registerStudent = async (req, res) => {
    try {
      const form = new formidable.IncomingForm();


      form.parse(req, async (err, fields, files) => {
        try {
          // Validate input fields
          const requiredFields = ['name', 'email', 'student_class', 'age', 'gender', 'guardian',
'guardian_phone', 'password'];
          const missingFields = requiredFields.filter(field => !fields[field]);


          if (missingFields.length > 0) {
            return res.status(400).json({
              success: false,
              message: `Missing required fields: ${missingFields.join(', ')}`
      });
}


// Process credentials
const email = fields.email[0].trim().toLowerCase();
const rawPassword = fields.password[0].trim();


if (!/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email)) {
      return res.status(400).json({
          success: false,
          message: "Invalid email format"
      });
}


if (rawPassword.length < 8) {
      return res.status(400).json({
          success: false,
          message: "Password must be at least 8 characters"
      });
}


// Check existing student
const existingStudent = await Student.findOne({
      email: { $regex: new RegExp(`^${email}$`, "i") }
});


if (existingStudent) {
      return res.status(409).json({
          success: false,
          message: "Email already registered"
      });
}
// In registerStudent and updateStudent
if (fields.student_class) {
      if (!mongoose.Types.ObjectId.isValid(fields.student_class[0])) {
          return res.status(400).json({
              success: false,
              message: "Invalid class ID format"
          });
      }
}


// Validate class ID
if (!mongoose.Types.ObjectId.isValid(fields.student_class[0])) {
      return res.status(400).json({
          success: false,
          message: "Invalid class ID format"
      });
}


// Handle image upload
const uploadDir = path.join(
      process.cwd(),
      process.env.STUDENT_IMAGE_PATH
);


if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir, { recursive: true, mode: 0o755 });
}


const studentImage = handleFileUpload(files, uploadDir, 'student_image');


// Create student entry
const hashedPassword = bcrypt.hashSync(rawPassword, 10);


const newStudent = new Student({
      school: req.user.schoolId,
      email: email,
      name: fields.name[0],
      student_class: fields.student_class[0], age: fields.age[0],
      gender: fields.gender[0],
      guardian: fields.guardian[0],
      guardian_phone: fields.guardian_phone[0],
      student_image: studentImage,
      password: hashedPassword
});
```

```
            const savedStudent = await newStudent.save();


            res.status(201).json({
                success: true,
                data: {
                    id: savedStudent._id,
                    email: savedStudent.email,
                    name: savedStudent.name
                },
                message: "Student registered successfully"
            });


        } catch (error) {
            console.error("Registration error:", error);


            if (error.code === 11000) {
                return res.status(400).json({
                    success: false,
                    message: "Email already exists in the system"
                });
            }


            res.status(500).json({
                success: false,
                message: "Registration failed",
                error: process.env.NODE_ENV === 'development' ? error.message : undefined
            });
        }
    });
  } catch (error) {
    res.status(500).json({
        success: false,
        message: "Server error",
        error: process.env.NODE_ENV === 'development' ? error.message : undefined
    });
  }
};
```

```
// Login student
exports.loginStudent = async (req, res) => {
  try {
    const { email, password } = req.body;


    if (!email || !password) {
      return res.status(400).json({
        success: false,
        message: "Email and password are required"
      });
    }


    const student = await Student.findOne({
      email: { $regex: new RegExp(`^${email.trim()}$`, 'i') }
    }).populate('school');


    if (!student) {
      return res.status(401).json({
        success: false,
        message: "Invalid credentials"
      });
    }


    const isMatch = await bcrypt.compare(password.trim(), student.password);
    if (!isMatch) {
      return res.status(401).json({
        success: false,
        message: "Invalid credentials"
      });
    }


    const payload = {
      id: student._id.toString(),
      schoolId: student.school._id.toString(),
      name: student.name,
      image_url: student.student_image,
      role: "STUDENT",
      email: student.email
    };
```

```
        const token = jwt.sign(
            payload,
            process.env.STUDENT_JWT_SECRET,
            { expiresIn: '7d', algorithm: 'HS256' }
        );


        res.status(200).json({
            success: true,
            message: "Login successful",
            token: token,
            user: payload
        });


    } catch (error) {
        console.error("Login error:", error);
        res.status(500).json({
            success: false,
            message: "Login failed",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Get all students (updated to populate class)
exports.getAllStudents = async (req, res) => {
    try {
        console.log('User making request:', req.user); // Debug log


        const filter = {};
        if (req.user.role === 'SCHOOL') {
            if (!req.user.schoolId) {
                return res.status(400).json({
                    success: false,
                    message: "School ID is missing in token"
                });
            }
            filter.school = req.user.schoolId;
        }
        console.log('Database query filter:', filter); // Debug lFog


        const students = await Student.find(filter)
            .populate('student_class', 'class_text')
            .select('-password -__v')
            .lean();


        console.log('Found students:', students.length); // Debug log


        res.status(200).json({
            success: true,
            message: "Students fetched successfully",
            count: students.length,
            data: students
        });
    } catch (error) {
        console.error("Get all students error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to fetch students",
            error: error.message // Always include error message in development
        });
    }
};
// Get single student data
exports.getStudentById = async (req, res) => {
    try {
        if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
            return res.status(400).json({
                success: false,
                message: "Invalid student ID format"
            });
        }


        const filter = { _id: req.params.id };
        if (req.user.role === 'SCHOOL') {
            filter.school = req.user.schoolId;
        }
        const student = await Student.findOne(filter)
            .populate('student_class', 'class_text') // Populate class with only class_text
            .select('-password')
            .lean();


        if (!student) {
            return res.status(404).json({
                success: false,
                message: "Student not found or unauthorized"
            });
        }


        res.status(200).json({
            success: true,
            data: student
        });


    } catch (error) {
        console.error("Get student error:", error);


        if (error.name === 'CastError') {
            return res.status(400).json({
                success: false,
                message: "Invalid student ID format"
            });
        }


        res.status(500).json({
            success: false,
            message: "Failed to fetch student data",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Update student details
```

```
exports.updateStudent = async (req, res) => {
    try {
        const form = new formidable.IncomingForm();


        form.parse(req, async (err, fields, files) => {
            try {
                let student;
                if (req.user.role === 'STUDENT') {
                    student = await Student.findById(req.user.id);
                } else if (req.user.role === 'SCHOOL') {
                    const studentId = fields.studentId?.[0] || req.user.id;
                    if (!studentId) {
                        return res.status(400).json({
                            success: false,
                            message: "Student ID is required for school users"
                        });
                    }
                    student = await Student.findOne({
                        _id: studentId,
                        school: req.user.schoolId
                    });
                } else {
                    return res.status(403).json({
                        success: false,
                        message: "Unauthorized access"
                    });
                }


                if (!student) {
                    return res.status(404).json({
                        success: false,
                        message: "Student not found"
                    });
                }
                // In registerStudent and updateStudent
                if (fields.student_class) {
                    if (!mongoose.Types.ObjectId.isValid(fields.student_class[0])) {
                        return res.status(400).json({
                            success: false,
                            message: "Invalid class ID format"
                        });
                    }
                }
                // Handle image update
                if (files?.student_image) {
                    const uploadDir = path.join(process.cwd(), process.env.STUDENT_IMAGE_PATH);


                    // Delete old image
                    if (student.student_image) {
                        const oldPath = path.join(uploadDir, student.student_image);
                        if (fs.existsSync(oldPath)) {
                            fs.unlinkSync(oldPath);
                        }
                    }


                    const studentImage = handleFileUpload(files, uploadDir, 'student_image');
                    student.student_image = studentImage;
                }


                // Update allowed fields
                const allowedUpdates = ['name', 'student_class', 'age', 'gender', 'guardian',
'guardian_phone'];
                allowedUpdates.forEach(field => {
                    if (fields[field]) {
                        // Special handling for student_class
                        if (field === 'student_class') {
                            if (mongoose.Types.ObjectId.isValid(fields[field][0])) {
                                student.student_class = fields.student_class[0];
                            }
                        } else {
                            student[field] = fields[field][0].trim();
                        }
                    }
                });


                // Handle password update
                if (fields.password) {
                    const newPassword = fields.password[0].trim();
                    if (newPassword.length < 8) {
                        return res.status(400).json({
                            success: false,
                            message: "Password must be at least 8 characters"
                        });
                    }
                    student.password = bcrypt.hashSync(newPassword, 10);
                }


                await student.save();


                // Populate class before returning
                const studentData = await Student.findById(student._id)
                    .populate('student_class', 'class_text')
                    .select('-password -__v')
                    .lean();


                res.status(200).json({
                    success: true,
                    message: "Student updated successfully",
                    data: studentData
                });


            } catch (error) {
                console.error("Update error:", error);
                res.status(500).json({
                    success: false,
                    message: "Failed to update student",
                    error: process.env.NODE_ENV === 'development' ? error.message : undefined
                });
            }
          });
         } catch (error) {
            console.error("Update student error:", error);
            res.status(500).json({
                success: false,
                message: "Internal server error",
                error: process.env.NODE_ENV === 'development' ? error.message : undefined
            });
         }
};
// Get students with query filters (search and class)
exports.getStudentsWithQuery = async (req, res) => {
         try {
        const filterQuery = { school: req.user.schoolId };


        if (req.query.search) {
            filterQuery['name'] = { $regex: req.query.search, $options: 'i' };
        }


        if (req.query.student_class) {
            if (mongoose.Types.ObjectId.isValid(req.query.student_class)) {
                filterQuery['student_class'] = new mongoose.Types.ObjectId(req.query.student_class);
            } else {
                return res.status(400).json({
                    success: false,
                    message: "Invalid class ID format"
                });
            }
        }


        const students = await Student.find(filterQuery)
            .populate('student_class', 'class_text')
            .select('-password');


        res.status(200).json({
            success: true,
            message: "Successfully fetched filtered students",
            data: students
        });
    } catch (error) {
        console.error("Filter students error:", error);
        res.status(500).json({
            success: false,
            message: "Internal Server Error",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Delete student by ID
exports.deleteStudentWithId = async (req, res) => {
    try {
```

```
    if (!mongoose.Types.ObjectId.isValid(req.params.id)) {
      return res.status(400).json({
        success: false,
        message: "Invalid student ID format"
      });
    }


    const student = await Student.findOneAndDelete({
      _id: req.params.id,
      school: req.user.schoolId
    });


    if (!student) {
      return res.status(404).json({
        success: false,
        message: "Student not found or not authorized to delete"
      });
    }


    // Delete associated image if exists
    if (student.student_image) {
      const imagePath = path.join(
        process.cwd(),
        process.env.STUDENT_IMAGE_PATH,
        student.student_image
      );


      if (fs.existsSync(imagePath)) {
        fs.unlinkSync(imagePath);
      }
    }


    res.status(200).json({
      success: true,
      message: "Student deleted successfully"
    });


  } catch (error) {
        console.error("Delete student error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to delete student",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};


// Get logged-in student's own data
exports.getStudentOwnData = async (req, res) => {
    try {
        const student = await Student.findById(req.user.id)
            .populate('student_class', 'class_text')
            .select('-password')
            .lean();


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


        res.status(200).json({
            success: true,
            data: student
        });


    } catch (error) {
        console.error("Get student data error:", error);
        res.status(500).json({
            success: false,
            message: "Failed to fetch student data",
            error: process.env.NODE_ENV === 'development' ? error.message : undefined
        });
    }
};
```

```
const express = require("express");
const authMiddleware = require("../auth/auth");
const { createClass, getAllClasses, updateClassWithId, deleteClassWithId } =
require("../controllers/class.controller");


const router = express.Router();


router.post("/create", authMiddleware(['SCHOOL']), createClass);
router.get("/all", authMiddleware(['SCHOOL','TEACHER','STUDENT']), getAllClasses);
router.patch("/update/:id", authMiddleware(['SCHOOL','TEACHER']), updateClassWithId); //
AUTHENTICATED USER FOR UPDATE
router.delete("/delete/:id", authMiddleware(['SCHOOL','TEACHER']), deleteClassWithId);


module.exports = router;



const mongoose = require('mongoose');


const classSchema = new mongoose.Schema({
  school: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'School',
    required: true
  },
  class_text: {
    type: String,
    required: true,
    trim: true
  },
  class_num: {
    type: Number,
    required: true,
    min: 1
  },
  class_teacher: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Teacher',
    default: null
  },
  createdAt: {
    type: Date,
    default: Date.now
  }
}, {
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});


// Virtual for students in this class
classSchema.virtual('students', {
  ref: 'Student',
  localField: '_id',
  foreignField: 'student_class',
  justOne: false
});


// Pre-remove hook to handle cleanup
classSchema.pre('remove', async function(next) {
  try {
    // Remove class reference from teachers
    await mongoose.model('Teacher').updateMany(
      { class_teacher_of: this._id },
      { $unset: { class_teacher_of: "" } }
    );
    next();
  } catch (error) {
    next(error);
  }
});


module.exports = mongoose.model("Class", classSchema);


const Class = require('../models/class.model');
const Student = require("../models/student.model");
const Exam = require("../models/examination.model");
const Schedule = require("../models/schedule.model");
```

```
module.exports = {


    getAllClasses: async (req, res) => {
        try {
            const schoolId = req.user.schoolId;
            const allClasses = await Class.find({ school: schoolId });
            res.status(200).json({ success: true, message: 'Success in fetching all Classes.', data:
allClasses });
        } catch (error) {
            console.log("getAllClasses error =>", error);
            res.status(500).json({ success: false, message: "Server Error in Getting Classes." });
        }
    },




    createClass: async (req, res) => {
        try {
            const newClass = new Class({
                school: req.user.schoolId,
                class_text: req.body.class_text,
                class_num: req.body.class_num
            });


            await newClass.save();
            res.status(200).json({ success: true, message: "Successfully created the class." })


        } catch (err) {
            res.status(500).json({ success: false, message: "Server Error in creating class." })


        }
    },
    // In your class controller when creating/updating a class
    updateClassWithId: async (req, res) => {
        try {
            let id = req.params.id;
            const { class_teacher } = req.body;
            // If updating class teacher, verify the teacher exists
            if (class_teacher) {
                const teacher = await Teacher.findById(class_teacher);
                if (!teacher) {
                    return res.status(400).json({ success: false, message: "Teacher not found" });
                }
            }


            await Class.findOneAndUpdate({ _id: id }, { $set: { ...req.body } });
            const classAfterUpdate = await Class.findOne({ _id: id });
            res.status(200).json({ success: true, message: 'Class Updated.', data: classAfterUpdate
});
        } catch (error) {
            console.log("Update class Error =>", error);
            res.status(500).json({ success: false, message: "Server Error in Updating Class." });
        }
    },
    deleteClassWithId: async (req, res) => {
        try {
            let id = req.params.id;
            let schoolId = req.user.schoolId;


            const classStudentCount = (await Student.find({ student_class: id, school: schoolId
})).length;
            const classExamCount = (await Exam.find({ class: id, school: schoolId })).length;
            const classScheduleCount = (await Schedule.find({ class: id, school: schoolId })).length;


            if ((classStudentCount == 0) && (classExamCount == 0) && (classScheduleCount == 0))
{
                await Class.findOneAndDelete({ _id: id, school: schoolId });
                res.status(200).json({ success: true, message: 'Class Deleted Successfully.' });
            } else {
                res.status(500).json({ success: false, message: 'This Class is already in use.' });
            }
        } catch (error) {
            console.log("Delete class Error =>", error);
            res.status(500).json({ success: false, message: "Server Error in Deleting Class." });
        }
    },
}



const jwt = require('jsonwebtoken');


module.exports = (roles = []) => {
  return async (req, res, next) => {
    try {
      // Get token from header
      const token = req.header('Authorization')?.replace('Bearer ', '');

      if (!token) {
        return res.status(401).json({
          success: false,
          message: "Authorization token required"
        });
      }


      // Decode token to get role before verification
      const decodedUnverified = jwt.decode(token);

      if (!decodedUnverified?.role) {
        return res.status(401).json({
          success: false,
          message: "Invalid token structure"
        });
      }


      // Get appropriate secret based on role
      const getSecret = () => {
        switch(decodedUnverified.role) {
          case 'SCHOOL':
            return process.env.SchoolJWT_SECRET;
          case 'TEACHER':
```

```
        return process.env.TEACHER_JWT_SECRET;
      case 'STUDENT':
        return process.env.STUDENT_JWT_SECRET;
      default:
        throw new Error('Invalid role type');
    }
  };


  const secret = getSecret();

  // Verify token with correct secret
  const decoded = jwt.verify(token, secret);


  // Verify role access
  if (roles.length > 0 && !roles.includes(decoded.role)) {
    return res.status(403).json({
      success: false,
      message: "Insufficient permissions"
    });
  }


  // Attach user data to request
  req.user = {
    id: decoded.id,
    schoolId: decoded.schoolId,
    role: decoded.role,
    email: decoded.email,
    name: decoded.name,
    image_url: decoded.image_url
  };


  next();
} catch (error) {
  console.error("Authentication error:", error);
  let message = "Invalid authentication token";

  if (error instanceof jwt.TokenExpiredError) {
    message = "Session expired. Please login again";
  } else if (error instanceof jwt.JsonWebTokenError) {
    message = "Invalid token format";
```

```
      } else if (error.message === 'Invalid role type') {
        message = "Unrecognized user role";
      }

      res.status(401).json({
        success: false,
        message: message,
        error: process.env.NODE_ENV === 'development' ? error.message : undefined
      });
    }
  };
};

const mongoose = require('mongoose');


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
}, {
  timestamps: true
});
```

```
module.exports = mongoose.model('Assignment', assignmentSchema);



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(
  '/create',
  auth(['TEACHER']),
  assignmentController.createAssignment
);


// Get assignments for student (STUDENT only)
router.get(
  '/student',
  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']),
  assignmentController.getSubmissionsForAssignment
);
```

```
module.exports = router;



const Assignment = require('../models/assignment.model');
const AssignmentSubmission = require('../models/submission.model');


// TEACHER creates an assignment
exports.createAssignment = async (req, res) => {
  try {
    const { title, description, subject, class: classId, dueDate, attachments } = req.body;


    const assignment = new Assignment({
      title,
      description,
      subject,
      class: classId,
      dueDate,
      attachments,
      school: req.user.schoolId,
      teacher: req.user.id
    });


    await assignment.save();


    res.status(201).json({ success: true, message: "Assignment created", data: assignment });
  } catch (error) {
    res.status(500).json({ success: false, message: "Error creating assignment", error:
error.message });
  }
};


// STUDENT fetches assignments for their class
exports.getAssignmentsForStudent = async (req, res) => {
  try {
    const { classId } = req.query;


    const assignments = await Assignment.find({ class: classId, school: req.user.schoolId })
```

```
      .populate('subject', 'subject_name')
      .populate('teacher', 'name');


    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 = async (req, res) => {
  try {
    const { assignmentId, fileUrl, remarks } = req.body;


    const submission = new AssignmentSubmission({
      assignment: assignmentId,
      student: req.user.id,
      fileUrl,
      remarks
    });


    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');


    res.status(200).json({ success: true, data: submissions });
  } catch (error) {
    res.status(500).json({ success: false, message: "Error fetching submissions", error:
error.message });
  }
};
```